Co-Design
of Structuring, Functionality, Distribution and Interactivity
for Information Systems

Collection of Recent Papers

Bernhard Thalheim
Christian-Albrechts-University Kiel, Department of Computer Science, 24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de

Table of contents

1. Co-design of structuring, functionality, distribution and interactivity
   published in [Tha04b]
   related papers: [Tha00b,ST09,Tha08b,Tha00a,Tha03,FJM+07,FJM+08]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

2. Achievements and Problems of Conceptual Modelling
   published in [Tha08a]
   related papers: [Tha00b,Tha07c,Tha07a,MNST07,FT07,BT07,Mor06,BST06]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

3. Methodik zum Co-Design von Informationssystemen
   published in [Tha08b]
   related papers: [Tha00b,Tha00a,Tha04b,Tha03,FJM+07,FJM+08]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

4. Towards ASM Engineering and Modelling
   published in [Tha00b,ST08a,PT03,BZT03,THa01,BZKST05,WST08,Tha00a,ST08f]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

5. The Conceptual Framework To User-Oriented Content Management
   published in [Tha06]
   related papers: [Tha00b,TV02,Tha04a]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

6. Engineering database component ware
   published in [Tha07b]
   related papers: [Tha00b,LT05,Tha02a,Tha05,FT03,ST04b,Tha02b,ST06a]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

7. Development of Collaboration Frameworks for Distributed Web Information Systems
   published in [ST07]
   related papers: [Tha07a,Tha08a,ST06b,ST04a,ST05,FRT05]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

8. Visual SQL: Towards ER-Based Object-Relational Database Querying
   published in [Tha08c]
   related papers: [Tha00b,JT03,TV02,Koc06,Ack06]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008
   published in [FJM+07]
   related papers: [Tha00b,JMTV05,FJM+08,FJM+09]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008
10. Co-Design of Web Information Systems Supported by SPICE
    published in [FJM+08,FJM+09]
    related papers: [Tha00b,JMTV05,FJM+07]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008
11. Quality Assurance in Web Information Systems Development
    published in [STZ]
    related papers: [Tha00b,JT05]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008
12. Capturing Forms in Web Information Systems Development
    published in [FRuDMT07]
    related papers: [Tha00b,ST09,NT08c,ST08c,ST08e]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008
13. Context analysis: Towards pragmatics of information system design
    published in [ST08b]
    related papers: [Tha00b,KSTZ03,BZKST04,FKST04,BZ04]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008
    published in [JTK+08]
    related papers: [Tha00b]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008
15. Databases of Personal Identifiable Information
    published in [AFT08]
    related papers: [Tha00b,AFFT05,TAFAS08]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008
16. Information Stream Based Model for Organizing Security
    published in [TAFAS08]
    related papers: [Tha00b,AFFT05,AFT08]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008
17. The Enhanced Entity-Relationship Model
    published in [Tha09]
    related papers: [Tha00b,07a,ST08a,ST08a,Tha07c,DMT07,MDT04]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008
18. Generalisation and specialisation
    published in [Tha09]
    related papers: [Tha00b,BT07]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008
19. Abstraction
    published in [Tha09]
    related papers: [Tha00b]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008

Please notice that the preprint [Tha03] (in German) can be downloaded from this website as well.
References


Co-Design of Structuring, Functionality, Distribution, and Interactivity for Information Systems

Bernhard Thalheim

Computer Science and Applied Mathematics Institute, University Kiel, Olshausenstrasse 40, 24098 Kiel, Germany
Email: thalheim@is.informatik.uni-kiel.de

Keywords: Co-design, database structure, information system functionality, interaction specification, distribution specification, entity-relationship model

Abstract

Database development has mainly been considered as development of database structuring. Functionality and interactivity specification has been neglected in the past. The derivability of functionality has been a reason for this restriction of the database design approach. At the same time, applications and the required functionality became more complex. Functionality specification may be based on workflow engines. Interaction support is often not specified but hidden within the interfaces. It may, however, be specified through the story space and the supporting media type suite. Distributed applications are based on an explicit specification of import/export views, on services provided and on exchange frames. The integration of all these parts has not yet been performed. The co-design approach aims in bridging all these different aspects of applications and provides additionally a sound methodology for development of all these aspects.

1 Introduction

1.1 Information Systems Design and Development

The problem of information system design can be stated as follows:

Design the logical and physical structure of an information system in a given database management system (or for a database paradigm), so that it contains all the information required by the user and required for the efficient behavior of the whole information system for all users. Furthermore, specify the database application processes and the user interaction.

The implicit goals of database design are:

- to meet all the information (contextual) requirements of the entire spectrum of users in a given application area;
- to provide a “natural” and easy-to-understand structuring of the information content;
- to preserve the designers entire semantic information for a later redesign;
- to achieve all the processing requirements and also a high degree of efficiency in processing;
- to achieve logical independence of query and transaction formulation on this level;
- to provide a simple and easily to comprehend user interface family.

Over the last years database structures have extensively been discussed. Almost all open questions have been satisfactorily solved. Modeling includes, however, more aspects:

Structuring of a database application is concerned with representing the database structure and the corresponding static integrity constraints.

Functionality of a database application is specified on the basis of processes and dynamic integrity constraints.

Distribution of information system components is specified through explicit specification of services and exchange frames.

Interactivity is provided by the system on the basis of foreseen stories for a number of envisioned actors and is based on media objects which are used to deliver the content of the database to users or to receive new content.

This understanding has led to the co-design approach to modeling by specification structuring, functionality, distribution, and interactivity. These four aspects of modeling have both syntactic and semantic elements.

1.2 Information System Models in General

Database design is based on one or more database models. Often, design is restricted to structural aspects. Static semantics which is based on static integrity constraints is sparsely used. Processes are then specified after implementing structures. Behavior of processes can be specified by dynamic integrity constraints. In a late stage, interfaces are developed. Due to this orientation the depth of the theoretical basis is different as shown in the following table displaying the state of the art in the 90ies:

<table>
<thead>
<tr>
<th>Component</th>
<th>Used in practice</th>
<th>Theoretical background</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structures</td>
<td>well done</td>
<td>well developed</td>
</tr>
<tr>
<td>Static semantics</td>
<td>partially used</td>
<td>well developed</td>
</tr>
<tr>
<td>Processes</td>
<td>somehow done</td>
<td>parts and pieces</td>
</tr>
<tr>
<td>Dynamic semantics</td>
<td>some parts</td>
<td>glimpses</td>
</tr>
<tr>
<td>Services</td>
<td>implementations</td>
<td>ad-hoc</td>
</tr>
<tr>
<td>Exchange frames</td>
<td>intentionally</td>
<td>nothing</td>
</tr>
<tr>
<td>Interfaces</td>
<td>intuitive</td>
<td>nothing</td>
</tr>
<tr>
<td>Stories</td>
<td>intuitive</td>
<td>nothing</td>
</tr>
</tbody>
</table>

Database design requires consistent and well-integrated development of structures, processes, distributions, and interfaces. We will demonstrate below that extended entity-relationship models allow to handle all four aspects.

## 2 Specification of Structuring

### 2.1 Languages for Structure Specification

Structuring of databases is based on three interleaved and dependent parts:

**Syntactics:** Inductive specification of structures uses a set of base types, a collection of constructors and an theory of construction limiting the application of constructors by rules or by formulas in deontic logics. In most cases, the theory may be dismissed. **Structural recursion** is the main specification vehicle.

**Semantics:** Specification of admissible databases on the basis of static integrity constraints describes those database states which are considered to be legal. If structural recursion is used then a variant of hierarchical first-order predicate logics may be used for description of integrity constraints.

**Pragmatics:** Description of context and intension is based either on explicit reference to the enterprise model, to enterprise tasks, to enterprize policy, and environments or on intensional logics used for relating the interpretation and meaning to users depending on time, location, and common sense.

The inductive specification of structuring is based on **base types** and **type constructors**. A **base type** is an algebraic structure \( B = (\text{Dom}(B), \text{Op}(B), \text{Pred}(B)) \) with a name, a set of values in a domain, a set of operations and a set of predicates. A class \( B^C \) on the base type is a collection of elements form \( \text{Dom}(B) \).

Usually, \( B^C \) is required to be set. It can be a list, a multi-set, a tree etc. Classes may be changed by applying operations. Elements of a class may be classified by the predicates.

A **type constructor** is a function from types to a new type. The constructor can be supplemented with a **selector** for retrieval (such as \( \text{Select} \)) and **update functions** (such as \( \text{Insert}, \text{Delete}, \text{and Update} \)) for value mapping from the new type to the component types or to the new type, with correctness criteria and rules for validation, with default rules, with one or more user representations, and with a physical representation or properties of the physical representation.

Typical constructors used for database definition are the **set**, **tuple**, **list** and **multiset** constructors. For instance, the set type is based on another type and uses algebra of operations such as union, intersection and complement. The retrieval function can be viewed in a straightforward manner as having a predicate parameter. The update functions such as \( \text{Insert}, \text{Delete} \) are defined as expressions of the set algebra.

The user representation is using the braces \{ , \}. The type constructors define type systems on basic data schemes, i.e. a collection of constructed data sets. In some database models, the type constructors are based on pointer semantics.

Other useful modeling constructs are **naming** and **referencing**. Each concept type and each concept class has a name. These names can be used for the definition of further types or referenced within a type definition. Often structures include also **optional components**. Optional components and references must be used with highest care since otherwise truly hyper-sophisticated logics such as topoi are required (Schewe 1994). A better approach to database modeling is the requirement of weak value-identifiability of all database objects (Schewe/Thalheim 1993).

### 2.2 Integrity Constraints

Integrity constraints are used to separate “good” states or sequences of states of a database system from those which are not intended. They are used for specification of semantics of both structures and processes. Therefore, consistency of database applications can not be treated without constraints. At the same time, constraints are given by users at various levels of abstraction, with a variety of vagueness and intensions behind and on the basis of different languages. For treatment and practical use, however, constraints must be specified in a clear and unequivocal form and language. In this case, we may translate these constraints to internal system procedures which are supporting consistency enforcement.

Each structure is also based on a set of **implicit model-inherent integrity constraints**:

- **Component-construction constraints** are based on existence, cardinality and inclusion of components. These constraints must be considered in the translation and implication process.

- **Identification constraints** are implicitly used for the set constructor. Each object either does not belong to a set or belongs only once to the set. Sets are based on simple generic functions. The identification property may be, however, only representable through automorphism groups (Beeri/Thalheim 1998). We shall later see that value-representability or weak-value representability lead to controllable structuring.

- A **cyclicity and finiteness** of structuring supports axiomatization and definition of the algebra. It must, however, be explicitly specified. Constraints such as cardinality constraints may be based on potential infinite cycles.

- **Superficial structuring** leads to representation of constraints through structures. In this case, implication of constraints is difficult to characterize.

- **Implicit model-inherent constraints** belong to the performance and maintenance traps. Integrity constraints can be specified based on the B(eeri)-V(aridi)-frame, i.e. by an implication with a formula for premises and a formula for the implication. BV-constraints do not lead to rigid limitation of expressibility. If structuring is hierarchical then BV-constraints can be specified within the first-order predicate logic. We may introduce a variety of different classes of integrity constraints defined:

- **Equality-generating constraints** allow to generate for a set of objects from one class or from several classes equalities among these objects or components of these objects.

- **Object-generating constraints** require the existence of another object set for a set of objects satisfying the premises.

A class \( C \) of integrity constraints is called **Hilbert-implication-closed** if it can be axiomatized by a finite set of bounded derivation rules and a finite set of axioms. It is well-known that the set of join dependencies is not Hilbert-implication-closed for relational structuring. However, an axiomatization exists with an unbounded rule, i.e. a rule with potentially infinite premises.
2.3 Representation Alternatives

The classical approach to database objects is to store an object based on strong typing. Each real life thing is thus represented by a number of objects which are either coupled by the object identifier or supported by specific maintenance procedures. In general, however, we might consider two different approaches to representation of objects:

Class-wise, identification-based representation: Things of reality may be represented by several objects. The object identifier (OID) supports identification without representing the complex real-life identification. Objects can be elements of several classes. In the early days of object-orientation it has been assumed that objects belong to one and only one class. This assumption has led to a number of migration problems which have not got any satisfying solution.

Structuring based on extended ER models (Thalheim 2000) or object-oriented database systems uses this option. Technology of relational and object-relational database systems is based on this representation alternative.

Object-wise representation: Graph-based models which have been developed in order to simplify the object-oriented approaches (Beeri/Thalheim 1998) display objects by their sub-graphs, i.e. by the set of nodes associated to a certain object and the corresponding edges. This representation corresponds to the representation used in standardization. XML is based on object-wise representation. It allows to use null values without notification. If a value for an object does not exist, is not known, is not applicable or cannot be obtained etc. the XML schema does not use the tag corresponding to the attribute or the component. Classes are hidden.

Object-wise representation has a high redundancy which must be maintained by the system thus decreasing performance to a significant extent. Beside the performance problems such systems also suffer from low scalability and bad utilization of resources. The operating of such systems leads to lock avalanches. Any modification of data requires a recursive lock of related objects.

For these reasons, objects-wise representation is applicable only under a number of restrictions:

- The application is stable and the data structures and the supporting basic functions necessary for the application are not changed during the life-span of the system.
- The data set is almost free of updates. Updates, insertions and deletions of data are only allowed in well-defined restricted ‘zones’ of the database.

A typical application area for object-wise storage are archiving systems, information presentation systems, and content management systems. They use an update system underneath. We call such systems play-out system. The data are stored in the way in which they are transferred to the user. The data modification system has a play-out generator that materializes all views necessary for the play-out system.

Other applications are main-memory databases without update. The SAP database system uses a huge set of related views.

We may use the first representation for our storage engine and the second representation for the input engine or the output engine in data warehouse approaches.

3 Specification of Functionality

3.1 Operations for Information Systems

General operations on type systems can be defined by structural recursion. Given types \( T, T' \) and a collection type \( C_T \) on \( T \) (e.g. set of values of type \( T \), bags, lists) and operations such as generalized union \( \cup_{CT} \), generalized intersection \( \cap_{CT} \), and generalized empty elements \( \emptyset_{CT} \) on \( C_T \). Given further an element \( h_0 \) on \( T' \) and two functions defined on the types \( h_1: T \to T' \) and \( h_2 : T' \times T' \to T' \).

We define the structural recursion by insert presentation for \( R^C \) on \( T \) as follows

\[
\begin{align*}
src_{h_0, h_1}(\emptyset_{CT}) &= h_0 \\
src_{h_0, h_1}(\langle s \rangle) &= h_1(s) \quad \text{for singleton collections } \langle s \rangle \\
src_{h_0, h_1}(\langle s \rangle \cup_{CT} \langle s' \rangle) &= h_2(h_1(s), src_{h_0, h_1}(\langle s' \rangle)) \\
& \quad \text{iff } \langle s \rangle \cap_{CT} R^C = \emptyset_{CT}.
\end{align*}
\]

All operations of the object-relational database model, the extended entity-relationship model and of other declarative database models can be defined by structural recursion, e.g.,

- selection is defined by \( src_{h_0, h_1}(\langle o \rangle) \) for the function \( \alpha \)

\[
\begin{align*}
\alpha_\alpha(\langle o \rangle) &= \begin{cases} \{ o \} & \text{if } | o | = \alpha \\ \emptyset & \text{otherwise} \end{cases}
\end{align*}
\]

- aggregation functions can be defined based on the two functions for null values

\[
\begin{align*}
h^0_f(s) &= \begin{cases} 0 & \text{if } s = \text{NULL} \\ f(s) & \text{if } s \neq \text{NULL} \end{cases} \\
h^{\text{undef}}_f(s) &= \begin{cases} \text{undef} & \text{if } s = \text{NULL} \\ f(s) & \text{if } s \neq \text{NULL} \end{cases}
\end{align*}
\]

through structural recursion, e.g.,

\[
\begin{align*}
\text{sum}^{\text{null}} &= src_{h_0, h_1} + \text{or} \\
\text{count}^{\text{null}} &= src_{h_0, h_1} + \text{or}
\end{align*}
\]

or the doubtful SQL definition of the average function

\[
\text{sum}^{\text{null}} \text{count}^{\text{null}}
\]

Similarly we may define intersection, union, difference, projection, join, nesting and un-nesting, renaming, insertion, deletion, and update.

Structural recursion is also limited in expressive power. Nondeterministic while tuple-generating programs (or object generating programs) cannot be expressed.

Operations may be either used for retrieval of values from the database or for state changes within the database.

The general frame for operation definition in the co-design approach is based on views used to restrict the scope, pre-, and postconditions used to restrict the applicability and the activation of operations and the explicit description of enforced operations:

- Operation \( \varphi \)

\[
[\text{View: } < \text{View_Name}> ] \\
[\text{Precondition: } < \text{Activation.Condition}> ] \\
[\text{Activated.Operation: } < \text{Specification}> ] \\
[\text{Postcondition: } < \text{Acceptance.Condition}> ] \\
[\text{Enforced.Operation: } < \text{Operation.Condition}> ]
\]

Operations defined on the basis of this general frame can be directly translated to database programs.
3.2 Dynamic Integrity Constraints

Database dynamics is defined on the basis of transition systems. A transition system on the schema $S$ is a pair

$$\mathcal{T} = \langle \mathcal{S}, \{\overset{\alpha}{\rightarrow} | \alpha \in \mathcal{L}\} \rangle$$

where $\mathcal{S}$ is a non-empty set of state variables, $\mathcal{L}$ is a non-empty set of labels, and $\overset{\alpha}{\rightarrow} \subseteq \mathcal{S} \times (\mathcal{S} \cup \{\infty\})$ for each $\alpha \in \mathcal{L}$. State variables are interpreted by states. Transitions are interpreted by transactions on $\mathcal{S}$. Database lifetime is specified on the basis of paths on $\mathcal{T}$. A path $\pi$ through a transition system is a finite or $\omega$ length sequence of the form $s_0 \overset{\alpha_1}{\rightarrow} s_1 \overset{\alpha_2}{\rightarrow} \ldots$. The length of a path is its number of transitions.

For the transition system $\mathcal{T}$ we can introduce now a temporal dynamic database logic using the quantifiers $\forall_f$ (always in the future), $\forall_p$ (always in the past), $\exists_f$ (sometimes in the future), $\exists_p$ (sometimes in the past).

First-order predicate logic can be extended on the basis of temporal operators. The validity function $I$ is extended by time. Assume a temporal class $(\mathcal{R}C, I_R)$. The validity function $I$ is extended by time and is defined on $S(t, S^\mathcal{R}, I_R)$. A formula $\alpha$ is valid for $I_{(\mathcal{R}C, I_R)}(t, s)$ if it is valid on the snapshot defined on $ts$, i.e. $I_{(\mathcal{R}C, I_R)}(\alpha, ts) = 1$ iff $I_{S(t, S^\mathcal{R}, I_R)}(\alpha, ts)$.

- For formulas without temporal prefix the extended validity function coincides with the usual validity function.
- $I(\forall_f \alpha, ts) = 1$ iff $I(\alpha, ts') = 1$ for all $ts' > ts$;
- $I(\forall_p \alpha, ts) = 1$ iff $I(\alpha, ts') = 1$ for all $ts' < ts$;
- $I(\exists_f \alpha, ts) = 1$ iff $I(\alpha, ts') = 1$ for some $ts' > ts$;
- $I(\exists_p \alpha, ts) = 1$ iff $I(\alpha, ts') = 1$ for some $ts' < ts$.

The modal operators $\forall_p$ and $\exists_p$ ($\forall_f$ and $\exists_f$ respectively) are dual operators, i.e. the two formulas $\forall_p \alpha$ and $\neg \exists_p \neg \alpha$ are equivalent. These operators can be mapped onto classical modal logic with the following definition:

- $\Box \alpha \equiv (\forall_f \alpha \land \forall_p \alpha \land \alpha)$;
- $\Diamond \alpha \equiv (\exists_f \alpha \lor \exists_p \alpha) \land \alpha$.

In addition, temporal operators until and next can be introduced.

The most important class of dynamic integrity constraint are state-transition constraints $\alpha \stackrel{O}{\rightarrow} \beta$ which use a pre-condition $\alpha$ and a post-condition $\beta$ for each operation $O$. The state-transition constraint $\alpha \stackrel{O}{\rightarrow} \beta$ can be expressed by the the temporal formula $\alpha \overset{O}{\rightarrow} \beta$.

Each finite set of static integrity constraints can be equivalently expressed by a set of state-transition constraints $\{\overset{\alpha \in \Sigma C}{\rightarrow} \land \overset{\alpha \in \Sigma C}{\rightarrow} | O \in Alg(M)\}$. Integrity constraints may be enforced

- either at the procedural level by application of
  - trigger constructs (Levene/Loizou 1999) in the so-called active event-condition-action setting,
  - greatest consistent specializations of operations (Schewe 1994),
  - or stored procedures, i.e., fully fledged programs considering all possible violations of integrity constraints,
- or at the transaction level by restricting sequences of state changes to those which do not violate integrity constraints,
- or by the DBMS on the basis of declarative specifications depending on the facilities of the DBMS,
- or at the interface level on the basis of consistent state changing operations.

3.3 Specification of Workflows

A large variety of approaches to workflow specification has been proposed in the literature. We prefer formal descriptions with graphical representations and thus avoid pitfalls of methods that are entirely based on graphical specification such as the and/or traps. We use basic computation step algebra introduced in (Thalheim/Düsterhöft 2001):

- **Basic control commands** are sequence $(\{\alpha\})$ (execute steps in sequence), parallel split $\parallel$ (execute steps in parallel), exclusive choice $\perp$ (choose one execution path from many alternatives), synchronization $\|\|$ (synchronize two parallel threads of execution by an synchronization condition $\|$), and simple merge $+$ (merge two alternative execution paths). The exclusive choice is considered to be the default parallel operation and is denoted by $\|$.\]

- **Structural control commands** are arbitrary cycles $\ast$ (execute steps w/out any structural restriction on loops), arbitrary cycles $\ast$ (execute steps w/out any structural restriction on loops but at least once), optional execution $\mid$ (execute the step zero times or once), implicit termination $\downarrow$ (terminate if there is nothing to be done), entry step in the step $\nearrow$ and termination step in the step $\searrow$.

The basic computation step algebra may be extended by **advanced step commands:**

- **Advanced branching and synchronization control commands** are multiple choice $\{\alpha\}$ (choose between $m$ and $n$ execution paths from several alternatives), multiple merge (merge many execution paths without synchronizing), discriminator (merge many execution paths without synchronizing, execute the subsequent steps only once), $n$-out-of-$m$ join (merge many execution paths, perform partial synchronization and execute subsequent step only once), and synchronizing join (merge many execution paths, synchronize if many paths are taken, simple merge if only one execution path is taken).

- We also may define **control commands on multiple objects** (CMO) such as CMO with a priori known design time knowledge (generate many instances of one step when a number of instances is known at the design time), CMO with a priori known runtime knowledge (generate many instances of one step when a number of instances can be determined at some point during the runtime (as in FOR loops)), CMO with no a priori known runtime knowledge (generate many instances of one step when a number of instances cannot be determined (as in a while loop)), and CMO requiring synchronization (synchronization edges) (generate many instances of one activity and synchronize afterwards).

- **State-based control commands** are deferred choice (execute one of the two alternative threads, the choice which thread is to be executed should be implicit), interleaved parallel executing (execute two activities in random order, but not in parallel), and milestone (enable an activity until a milestone has been reached).
Finally, cancellation control commands are used, e.g., cancel step (cancel (disable) an enabled step) and cancel case (cancel (disable) the case).

These control composition operators are generalizations of workflow patterns and follow approaches developed for Petri net algebras.

### 3.4 Architecture of Database Engines

Operating of information systems is modeled by separating the systems into four state spaces:

\[ \mathcal{E}R_c = (\text{input states } I_N, \text{output states } O(MT), \text{engine states } DBMS, \text{database states } DB) \]

The input states accommodate the input to the database system, i.e., queries and data. The output space allow to model the output of the DBMS, i.e., output data of the engine and error messages. The internal state space of the engine is represented by engine states. The database content of the database system is represented in the database states. The four state spaces can be structured. This structuring is reflected in all four state spaces. For instance, if the database states are structured by a database schema then the input states are accordingly structured.

Using value-based or object-relational models the database states are structured by a database schema. A set of state changes rules is in this case a change to one of the relations. State changes are modeled on the basis of abstract state machines (Böger/Stäck 2003) through state change rules. An engine is specified by its programs and its control. We follow this approach and distinguish between

- programs that specify units of work or services and meet service quality obligations and

- control and coordination that is specified on the level of program blocks with or without atomicity and consistency requirements or specified through job control commands.

Programs are called with instantiated parameters for their variables. Variables are either static or stack or explicit or implicit variables. We may use furthermore call parameters such as onSubmit and presentationMode, priority parameters such as onFocus and emphasisMode, control parameters such as onRecovery and hookOnProcess, error parameter such as onError and notifyMode, and finally general transfer parameters such as onReceive and validUntil.

Atomicity and consistency requirements are supported by the variety of transaction models. Typical examples are flat transactions, sagas, join-and-split transactions, contracts or long running activities (Thalheim 2000).

State changes \( T(s_1, ..., s_n) \) := \( t \) of a sub-type \( T \) of the database engine \( \mathcal{E}R_c \). A set \( U = \{ T_i(s_{i,1}, ..., s_{i,n}) : o_i \mid 1 \leq i \leq m \} \) of object-based state changes is consistent, if the equality \( o_i = o_j \) is implied by \( T_i(s_{i,1}, ..., s_{i,n}) = T_j(s_{j,1}, ..., s_{j,n}) \) for \( 1 \leq i < j \leq m \).

The result of an execution of a consistent set \( U \) of state changes leads to a new state \( \mathcal{E}R_c + U \)

\[
(\mathcal{E}R_c + U)(o) = \begin{cases} 
\text{Update}(T_i(s_{i,1}, ..., s_{i,n}), o_i) \\
\text{otherwise} 
\end{cases}
\]

for objects \( o \) of \( \mathcal{E}R_c \).

A parameterizable program \( r(x_1, ..., x_n) = P \) of arity \( n \) consists of a program name \( r \), a transition rule \( P \) and a set \( \{x_1, ..., x_n\} \) of free variables of \( P \).

An information system \( \mathcal{E}R^C_c \) is a model of \( \phi \) (\( \mathcal{E}R^C_c \models \phi \)) if \( \phi \text{\langle E\rangle}^C = true \) for all variable assignments \( \zeta \) for free variables of \( \phi \).

Two typical program constructors are the execution of a program for all values that satisfy a certain restriction

\[
\text{FOR ALL } x \text{ WITH } \phi \rightarrow \text{DO } P
\]

and the repetition of a program step in a loop

\[
\text{DO } P_i \text{ LOOP } \alpha
\]

We introduce also other program constructors such as sequential execution, branch, parallel execution, execution after value assignment, execution after choosing an arbitrary value, skip, modification of a information system state, and call of a subprogram.

We use the abstract state machine approach also for definition of semantics of the programs.

A transition rule \( P \) leads to a set \( U \) of state changing operations in a state \( \mathcal{E}R^C \) if it is consistent. The state of the information system is changed for a variable assignment \( \zeta \) to \( \text{yields}(P, \mathcal{E}R^C, \zeta, U) \).

Semantics of transition rules is defined in a calculus that uses rules of the form

\[
\text{prerequisite}_1, ..., \text{prerequisite}_n \rightarrow \text{conclusion}
\]

For instance, the state change imposed by the first program step is defined by

\[
\forall a \in I : \text{yields}(P, \mathcal{E}R^C, \zeta[x \mapsto a], U_a)
\]

\[
\text{yields}(\text{FOR ALL } x \text{ WITH } \phi \rightarrow \text{DO } P, \mathcal{E}R^C, \zeta, \bigcup_{a \in I} U_a)
\]

where \( I = \text{range}(x, \phi, \mathcal{E}R^C, \zeta) \)

The range \( \text{range}(x, \phi, \mathcal{E}R^C, \zeta) \) is defined by the set \( \{ o \in \mathcal{E}R^C \mid [\phi]_{\langle E\rangle}^C(o) = true \} \).

### 4 Specification of Distribution

Specification of distribution has neglected over a long period. Instead of explicit specification of distribution different collaborating approaches have been tried such as multi-database systems, federated database systems,

#### 4.1 View Suite

Classically, (simple) views are defined as singleton types which data is collected from the database by some query.

\[
\text{create view name (projection variables)} \rightarrow \text{select projection expression from Database sub-schema where selection condition group by expression for grouping having selection among groups order by order within the view}
\]

Since we may have decided to use the class-wise representation simple views are not the most appropriate structure for exchange specification. Instead we use view suites for exchange. A suite consists of a set of elements, an integration or association schema and obligations requiring maintenance of the association.
Simple examples of a view suites are already discussed in (Thalheim 2000) where view suites are ER schemata. The integration is given by the schema. Obligations are based on the master-slave paradigm, i.e., the state of the view suite classes is changed whenever an appropriate part of the database is changed.

Additionally, views should support services. Services provide their own data and functionality. This object-orientation is a useful approach whenever data should be used without direct or remote connection to the database engine.

We generalize the view specification frame used in relational databases by the frame:

\[
\text{generate Mapping} : \ Vars \rightarrow \text{OUTPUT STRUCTURE} \\
\text{from DATABASE TYPES} \\
\text{where SELECTION CONDITION} \\
\text{represent using GENERAL PRESENTATION STYLE} \\
\& \ \text{ABSTRATION (GRANULARITY, MEASURE, PRECISION)} \\
\& \ \text{ORDERS WITHIN THE PRESENTATION} \\
\& \ \text{HIERARCHICAL REPRESENTATIONS} \\
\& \ \text{POINTS OF VIEW} \\
\& \ \text{SEPARATION} \\
\text{browsing definition condition} \\
\& \ \text{NAVIGATION} \\
\text{functions SEARCH FUNCTIONS} \\
\& \ \text{EXPORT FUNCTIONS} \\
\& \ \text{INPUT FUNCTIONS} \\
\& \ \text{SESSION FUNCTIONS} \\
\& \ \text{MARKING FUNCTIONS} \\
\]

The extension of views by functions seems to be an overhead during database design. Since we extensively use views in distributed environments we save efforts of parallel and repetitive development due to the development of the entire view suite instead of developing each view by its own.

### 4.2 Services

Services are usually investigating on one of the (seven) layers of communication systems. They are characterized by two parameters: Functionality and quality of service. Nowadays we prefer a more modern approach (Lockemann 2003). Instead of functions we consider informational processes. Quality of service is bounded by a number of properties that are stated either at implementation layer or at conceptual layer or at business user layer.

Services consist of informational processes, the characteristics provided and properties guaranteeing service quality, i.e., \( \mathcal{S} = (\mathcal{I}, \mathcal{F}, \Sigma_{\mathcal{S}}) \) where \( \mathcal{I} = (\mathcal{V}, \mathcal{M}, \Sigma_{\mathcal{I}}) \).

Informational processes are specified by the ingredients:

#### Views from the view suite \( \mathcal{V} \) are the resources for informational processes. Since views are extended by functions they are computational and may be used as statistical packages, data warehouses or data mining algorithms.

The services manager \( \mathcal{M} \) supports functionality and quality of services and manages containers, their play-out and their delivery to the client. It is referred to as a service provider.

The competence of a service manifests itself in the set of tasks \( \mathcal{T} \) that may be performed.

Service characteristics \( \mathcal{F} \) is characterized depending on the abstraction layers:

#### Service characteristics at conceptual layer describe properties the service must provide in order to meet the service level agreements. Further, functions available to the client at specified by their interfaces and semantic effects.

#### Service characteristics at implementation layer specify the syntactical interfaces of functions, the data sets provided and their behavior and constraints to the information system and to the client.

Quality of service \( \Sigma_{\mathcal{S}} \) is characterized depending on the abstraction layers:

- Quality parameters at business user layer may include **ubiquity** (access unrestricted in time and space) and **security** (against failures, attacks, errors; trustworthy).
- Quality parameters at conceptual layer subsume **interpretability** (formal framework for interpretation) and **consistency** (of data and functions).
- Quality parameters at implementation layer include **durability** (access to the entire information unless it is explicitly overwritten), **robustness** (based on a failure model for resilience, conflicts, and persistency), **performance** (depending on the cost model, response time and throughput), and **scalability** (to changes in services, number of clients and servers).

### 4.3 Exchange Frames

The exchange frame is defined by exchange architecture usually provided a system architecture integrating the information systems through communication and exchange systems, collaboration style specifying the supporting programs, the style of cooperation and the coordination facilities, and collaboration pattern specifying the roles of the partners, their responsibilities, their rights and the protocols they may rely on.

![Figure 1: Generalization of the Three-Level Architecture to Distributed Schema](image-url)
Distributed database systems are based on local database systems and follow a certain integration strategy. Integration is based on total integration of the local conceptual schemata into a global distribution schema. The architecture is displayed in Figure 1.

Beside the classical distributed system we support also other architecture such as database farms, incremental information system societies and cooperating information systems. The later are based on the concept of cooperating views (Thalheim 2000). Incremental information system societies are the basis for facility management systems. Simple incremental information systems are data warehouses and content management systems.

Database farms are generalizing and extending the approaches to federated information systems and mediators. Their architecture is displayed in Figure 2. Farms are based on the co-design approach and the information unit and container paradigm:

Information units are generalized views. Views are generated on the basis of the database. Units are views extended by functionality necessary for the utilization of view data. We distinguish between retrieval information units and modification information units. The first are used for data injection. The later allow to modify the local database.

Containers support the export and the import of data by bundling information units provided by view states. Units are composed to containers which can be loaded and unloaded in a specific way. The unloading procedure supports the dialogue scenes and steps.

The global communication and farming system provides the exchange protocols, has facilities for loading and unloading containers and for modification of modification information units.

We do not want to integrate entirely the local databases but provide only cooperating views.

The exchange architecture may include the workplace of the client describing the actors, groups, roles and rights of actors within a group, the task portfolio and the organization of the collaboration, communication, and cooperation.

The collaboration style is based on four components describing supporting programs of the information system including session management, user management, and payment or billing systems;

data access pattern for data release through the net, e.g., broadcast or P2P; for sharing of resources either based on transaction, consensus, and recovery models or based on replication with fault management, and for remote access including scheduling of access;

the style of collaboration on the basis of peer-to-peer models or component models or push-event models which restrict possible communication;

and the coordination workflows describing the interplay among partners, discourse types, name space mappings, and rules for collaboration.

We know a number of collaboration pattern supporting access and configuration (wrapper facade, component configuration, interceptor, extension interface), event processing (reactor, proactor, asynchronous completion token, accept connector), synchronization (scoped locking, strategized locking, thread-safe interface, double-checked locking optimization) and parallel execution (active object, monitor object, half-sync/half-async, leader/followers, thread-specific storage):

Proxy collaboration uses partial system copies (remote proxy, protection proxy, cache proxy, synchronization proxy, etc.).

Broker collaboration supports coordination of communication either directly, through message passing, based on trading paradigms, by adapter-broker systems, or callback-broker systems.

Master/slave collaboration uses tight replication in various application scenarios (fault tolerance, parallel execution, precision improvement; as processes, threads; with(out) coordination).

Client/dispatcher collaboration is based on name spaces and mappings.

Publisher/subscriber collaboration is also known as the observer-dependents paradigm. It may use active subscribers or passive ones. Subscribes have their subscription profile.

Model/view/controller collaboration is similar to the three-layer architecture of database systems. Views and controllers define the interfaces.

Collaboration pattern generalize protocols. They include the description of partners, their responsibilities, roles and rights.
5 Specification of Interactivity

Interactivity of information systems has been mainly considered on the level of presentation systems by the Arch or Seeheim separation between the application system and the presentation system. Structuring and functionality are specified within a database modeling language and its corresponding algebra. Pragmatics is usually not considered within the database model. The interaction with the application system is based on a set of views which are defined on the database structure and are supported by some functionality. In the co-design framework we generalize this approach by

introduction of media objects which are generalized views that have been extended by functionality necessary, are adapted to the users needs and delivered to the actor by a container (Schewe/Thalheim 2000) and by

introduction of story spaces (Srinivasa 2001) which specify the stories of usage by groups of users (called actors) in their context, can be specialized to the actual scenario of usage and use a variety of play-out facilities.

User interaction modeling involves several partners (grouped according to characteristics; group representatives are called ‘actors’), manifests itself in diverse activities and creates an interplay between these activities. Interaction modeling includes modeling of environments, tasks and actors beside modeling of interaction flow, interaction content and interaction form.

The general architecture of a web information system is shown in Figure 3. This architecture has successfully been applied in more than 30 projects resulting in huge or very large information-intensive websites and in more than 100 projects aiming in building large information systems.

![Interactive Information System](image)

**Figure 3: Specification Approach for Information Systems**

5.1 Story Space

Modeling of interaction must support multiple scenarios. In this case, user profiles, user portfolios, and the user environment must be taken into consideration. The story of interaction is the intrigue or plot of a narrative work or an account of events. The language SiteLang (Thalheim/Düsterhöft 2001) offers concepts and notation for specification of story spaces, scene and scenarios in them.

Within a story one can distinguish threads of activity, so-called scenarios, i.e., paths of scenes that are connected by transitions. We define the story space $\Sigma_W$ as the 7-tuple $(S_W, T_W, E_W, G_W, A_W, A_W, \kappa_W)$ where $S_W, T_W, E_W, G_W$ and $A_W$ are the set of scenes created by $W$, the set of scene transitions and events that can occur, the set of guards and the set of actions that are relevant for $W$, respectively. Thus, $T_W$ is a subset of $S_W \times S_W$. Furthermore $\lambda_W : S_W \rightarrow \text{SceneSpec}$ is a function associating a scene specification with each scene in $S_W$, and $\kappa_W : T_W \rightarrow E_W \times G_W \times A_W$. $t \mapsto (e, g, a)$ is a function associating with each scene transition $t$ occurring in $W$ the event $e$ that triggers transition $t$, the guard $g$, i.e., a logical condition blocking the transition if it evaluates to false on occurrence of $e$, and the action $a$ that is performed while the transition takes place.

We consider scenes as the conceptual locations at which the interaction, i.e., dialogue takes place. Dialogues can be specified using so-called dialogue-step expressions. Scenes can be distinguished from each other by means of their identifier: Scene-ID. With each scene there is associated a media object and the set of actors that are involved in it. Furthermore, with each scene a representation specification is associated as well as a context. Scenes therefore can be specified using the following frame:

$\text{Scene} = (\text{Scene-ID}, \text{DialogueStepExpression}, \text{Data views with associated functions}, \text{User}, \text{UserID}, \text{UserRight}, \text{UserTasksAssigned}, \text{UserRoles}, \text{Representation (styles, defaults, emphasis, ...)}, \text{Context (equipment, channel, particular)})$

Dialogue-step expressions consist of dialogues and operators applied to them. A typical scene is displayed in Figure 4. A learner may submit solutions in the data mining cup. Before doing so, the user must pay a certain fee if she has not already been paying. The system either knows the user and his/her profile. If the user has already paid the fee then the payment dialogue step is not shown. If the user has not paid the fee or is an anonymous user then the fee dialogue step must be visited and the dialogue step for task completion is achievable only after payment.
5.2 Media Type Suite

Media types have been introduced in (Schewe/Thalheim 2000). Since users have very different needs in data depending on their work history, their portfolio, their profile and their environment we send the data packed into containers. Containers have the full functionality of the view suite. Media type suites are based on view suites and use a special delivery and extraction facility. The media type suite is managed by a system consisting of three components:

**Media object extraction system:** Media objects are extracted and purged from database, information or knowledge base systems and summarized and compiled into media objects. Media objects have a structuring and a functionality which allows to use these in a variety of ways depending on the current task.

**Media object storage and retrieval system:** Media objects can be generated on the fly whenever we need the content or can be stored in the storage and retrieval subsystem. Since their generation is usually complex and a variety of versions must be kept, we store these media objects in the subsystem.

**Media object delivery system:** Media objects are used in a large variety of tasks, by a large variety of users in various social and organizational contexts and further in various environments. We use a media object delivery system for delivering data to the user in form the user has requested. Containers contain and manage the set of media object that are delivered to one user. The user receives the user-adapted container and may use this container as the desktop database.

This understanding closely follows the data warehouse paradigm. It is also based on the classical model-view-control paradigm. We generalize this paradigm to media objects, which may be viewed in a large variety of ways and which can be generated and controlled by generators.

6 Integrating Specification Aspects into Co-Design

The languages introduced so far seem to be rather complex and the consistent development of all aspects of information systems seems to be rather difficult. We developed a number of methodologies to development in order to overcome difficulties in consistent and complete development. Most of them are based on top-down or refinement approaches that separate aspects of concern into abstraction layers and that use extension, detailisation, restructuring as refinement operations.

6.1 The Abstraction Layer Model for Information Systems Development

We observe that information systems are specified at different abstraction layers:

1. The **motivation layer** addresses the purpose of the information system, i.e. its mission statement and the anticipated customer types including their goals. The results of the design process are conducted into the **stakeholder contract specification**.

2. The **strategic layer** describes the information system, analyzes business processes and aims in elicitation of the requirements to the information system. The results of the design process are combined into the system specification.

3. The **business layer** deals with modelling the anticipated usage of the information system in terms of customer types, locations of the information space, transitions between them, and dialogues and discourses between categories of users (called actors). The result of this abstraction layer is compiled into an extended system manual including mockups of the interfaces and scenarios of utilization.

4. The **conceptual layer** integrates the conceptual specification of structuring, functionality, distribution and interactivity. The results of this step are the database schema, the workflows, the view and media type suites, the specification of services and exchange frames, and the story space.

5. At the **implementation layer**, logical and physical database structures, integrity enforcement procedures, programs, and interfaces are specified within the language framework of the intended platform. The result of specification at the implementation layer is the implementation model. This model is influenced by the builder of the information system.

6. The **exploitation layer** is not considered here. Maintenance, education, introduction and administration are usually out of the scope of conceptualization of an application.

![Figure 5: The Abstraction Layer Model of the Database Design Process](image)

6.2 The Co-Design Methodology

Methodologies must confirm both with the SPICE v. 2.0 and SW-CMM v. 2.0 requirements for consistent system development. The co-design framework is based on a step-wise refinement along the abstraction layers. Since the four aspects of information systems - structuring, functionality, distribution and interactivity - are interrelated they cannot be developed separately. The methodology sketched below is based on steps in the following specification frame:

<table>
<thead>
<tr>
<th>Rule #</th>
<th>Task 1</th>
<th>Task 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Name of the step</td>
<td>Documents of previous steps</td>
<td>(IS development documents)</td>
</tr>
<tr>
<td>2. Used documents</td>
<td>Customer documents and information</td>
<td></td>
</tr>
</tbody>
</table>
The steps used in one of the methodologies are:

**Motivation layer**
1. Developing visions, aims and goals
2. Analysis of challenges and competitors

**Strategic layer**
3. Separation into system components
4. Sketching the story space
5. Sketching the view suite
6. Specifying business processes

**Business user layer**
7. Development of scenarios of the story space
8. Elicitation of main data types and their associations
9. Development of kernel integrity constraints, e.g., identification constraints
10. Specification of user actions, usability requirements, and sketching media types
11. Elicitation of ubiquity and security requirements

**Conceptual layer**
12. Specification of the story space
13. Development of data types, integrity constraint, their enforcement
14. Specification of the view suite, services and exchange frames
15. Development of workflows
16. Control of results by sample data, sample processes, and sample scenarios
17. Specification of the media type suite
18. Modular refinement of types, views, operations, services, and scenes
19. Normalization of structures
20. Integration of components along architecture

**Implementation layer**
21. Transformation of conceptual schemata into logical schemata, programs, and interfaces
22. Development of logical services and exchange frames

23. Developing solutions for performance improvement, tuning
24. Transformation of logical schemata into physical schemata
25. Checking durability, robustness, scalability, and extensibility

**Concluding Remark**
The co-design methodology has been practically applied in a large number of information system projects and has nevertheless a sound theoretical basis. We do not want to compete with UML but support system development at a sound basis without ambiguity, ellipses and conceptual mismatches.

**References**


Remark: Our main aim has been the presentation of the co-design framework. We restrict thus the bibliography only to those references which are necessary for this paper. An extensive bibliography on relevant literature in this field can be found in (Thalheim 2000).

Acknowledgement. I want to thank Hans-Joachim Klein for his comments and discussions.
Achievements and Problems of Conceptual Modelling

Bernhard Thalheim
Department of Computer Science,
Christian Albrechts University Kiel, 24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de

Abstract. Database and information systems technology has substantially changed. Nowadays, content management systems, (information-intensive) web services, collaborating systems, internet databases, OLAP databases etc. have become buzzwords. At the same time, object-relational technology has gained the maturity for being widely applied. Conceptual modelling has not (yet) covered all these novel topics. It has been concentrated for more than two decades around specification of structures. Meanwhile, functionality, interactivity and distribution must be included into conceptual modelling of information systems. Also, some of the open problems that have been already discussed in 1987 [15][16] still remain to be open. At the same time, novel models such as object-relational models or XML-based models have been developed. They did not overcome all the problems but have been sharpening and extending the variety of open problems. The open problem presented are given for classical areas of database research, i.e., structuring and functionality. The entire are of distribution and interaction is currently an area of very intensive research.

The presentation of open problems is combined with the introduction to the achievements of conceptual modelling. The paper develops an approach to conceptual modelling for object-relational, collaborating information systems that support virtual communities of work, integration of information systems, varieties of architecture such as the OLTP-OLAP architecture, varieties of play-out and play-in systems, and data analysis engines. The paper is based on an extended entity-relationship model that covers all structuring facilities of object-relational systems. It uses the theory of media types and storyboards for the specification of interactivity and provides a framework for collaboration.

The paper presents 20 open problems that need to be solved for conceptual modelling. The problems are sketched. Main references and the background are given. Additional references can be provided by the author on demand.

1 Introduction

1.1 Information Systems Design and Development

The problem of information system[1] design can be stated as follows:

Design the logical and physical structure of an information system in a given database management system (or for a database paradigm), so that it contains

---

1 A database system consists of a number of databases and a database management system. An information system extends the database system by the application system and by presentation systems.

© Springer-Verlag Berlin Heidelberg 2007
Achievements and Problems of Conceptual Modelling

all the information required by the user and required for the efficient behavior of the whole information system for all users. Furthermore, specify the database application processes and the user interaction.

The implicit goals of database design are:

- to meet all the information (contextual) requirements of the entire spectrum of users in a given application area;
- to provide a “natural” and easy-to-understand structuring of the information content;
- to preserve the designers entire semantic information for a later redesign;
- to achieve all the processing requirements and also a high degree of efficiency in processing;
- to achieve logical independence of query and transaction formulation on this level;
- to provide a simple and easily to comprehend user interface family.

Over the last years database structures have extensively been discussed. Some of the open questions have been satisfactorily solved. Modelling includes, however, more aspects:

**Structuring** of a database application is concerned with representing the database structure and the corresponding static integrity constraints.

**Functionality** of a database application is specified on the basis of processes and dynamic integrity constraints.

**Distribution** of information system components is specified through explicit specification of distribution.

**Interactivity** is provided by the system on the basis of foreseen stories for a number of envisioned actors and is based on media objects which are used to deliver the content of the database to users or to receive new content.

This understanding has led to the **co-design approach** to modelling by specification structuring, functionality, distribution, and interactivity. These four aspects of modelling have both syntactic and semantic elements.

Nevertheless, the main open problem has not yet been solved:

**Open problem 1.**

Find a common motivation, a common formal model and a correspondence that justify the properties and formalize the characteristics.

### 1.2 Information System Models in General

Database design is based on one or more database models. Often, design is restricted to structural aspects. Static semantics which is based on static integrity constraints is sparsely used. Processes are then specified after implementing structures. Behavior of processes can be specified by dynamic integrity constraints. In a late stage, interfaces are developed. Due to this orientation the depth of the theoretical basis is different as shown in the following table displaying the state of the art in the 90ies:
Database design requires consistent and well-integrated development of structures, processes, distribution, and interfaces. We will demonstrate below that extended entity-relationship models allow to handle all four aspects.

Database systems are now extended to web information systems, to data warehouse, to intelligent knowledge bases, and to data analysis systems. This extension can be developed in a conservative fashion or based on novel paradigms. As long as novel paradigms do not overcome the problematic parts of database systems operating, conservative extension must be preferred. In this case we need a good architecture \cite{9,7} for extensions of systems.

**Open problem 2.**

Find an architecture for generic extension of database systems that entirely supports the extensions, that allows to model all facilities and that supports reasoning on system properties.

At the same time, we need to model database systems quality. The quality criteria are often stated in a rather fuzzy form. Typical quality criteria are \cite{4} accuracy, changeability, fault tolerance, operability, performance, privacy, recoverability, reliability, resource efficiency, safety, security, stability, and testability.

**Open problem 3.**

Define in a formal form quality criteria, quality attributes, and quality metrics for conceptual modelling and develop a framework for their enforcement, control, and refinement.

## 2 Towards a Science of Modelling

### 2.1 The Documentation and Knowledge Gap in Information Systems Modelling

Managing information systems applications becomes crucial for evolution and prosperity of companies. Information systems are used over decades whereas software and hardware changes more often. For this reason, the complete information systems development process must be well-documented. The entire development knowledge that is necessary for building, using, and maintaining information systems must be kept over the lifetime of the system. The documentation includes meta-data on the enterprise system, the technical process supporting the system, the business processes supported by
the system, services and software tools used within the system, organizational policies and people.

Classical information systems development follows often the late documentation approach, i.e. the development documentation is based on the final result of the conceptualization, i.e. consists of the conceptual schema and its translation to relational or object-relational system languages. Already [5] discussed the forgetful development of software products. Classically we observe that

- developers base their design decisions on a "partial reality", i.e. on a number of observed properties within a part of the application,
- developers are developing the information system within a certain context,
- developers reuse their experience gained in former projects and solutions known for their reference models, and
- developers use a number of theories with a certain exactness and rigidity.

The design decisions made during the design process are deeply influenced by these four hidden factors. In some approaches revisions made during the information systems development are recorded. However, since the background knowledge is not recorded
the documentation of the information systems development is fragmentary. This knowledge gap is visualized in Figure 1 [18].

The most pressing challenges that organizations are currently facing are: provide IT portfolio management, reduce IT redundancy, prevent IT applications failure, reduce IT expenditures, enable knowledge management, adhere to regulatory requirements, and enable enterprise-wide integration of applications.

Open problem 4.

Develop a specification language that allows to specify the modelling decision together with the causes for choosing the given decision and that allows to reason on consistency of causes.

2.2 Modelling in General and for Information Systems

Conceptual modelling and modelling differ a lot. Mathematical modelling satisfies three requirements based on a chosen modelling language [21]:

- adequate and approximate representation $R$ of phenomena $P$ observed in reality;
- unique representation of these phenomena without any choice for a representation style;
- repetitiveness of the result of modelling whenever modelling is repeated.

The process of modelling thus includes the following steps:

1. formulation of laws, general properties, associations etc. within the chosen modelling language and statement of problems and requirements under consideration;
2. solution of the problems and development of a theory for the solution based on methods developed so far;
3. check whether the models satisfies all requirements of practical exploitation and development of refined models;
4. development of theories on the practical application based on the model.

Conceptual modelling considers mainly the first process step. The second step is often neglected. The third step comes into play whenever practical exploitation leads to a

---

2 Due to our involvement into the development and the service for the CASE workbenchs (DB) and ID we have collected a large number of real life applications. Some of them have been really large or very large, i.e., consisting of more than 1.000 attribute, entity and relationship types. The largest schema in our database schema library contains of more than 19.000 entity and relationship types and more than 60.000 attribute types that needs to be considered as different. Another large database schema is the SAP R/3 schema. It has been analyzed in 1999 by a SAP group headed by the third author during his sabbatical at SAP. At that time, the R/3 database used more than 16.500 relation types, more than 35.000 views and more than 150.000 functions. The number of attributes has been estimated by 40.000. Meanwhile, more than 21.000 relation types are used. The schema has a large number of redundant types which redundancy is only partially maintained. The SAP R/3 is a very typical example of a poorly documented system. Most of the design decisions are now forgotten. The high type redundancy is mainly caused by the incomplete knowledge on the schema that has been developed in different departments of SAP.
change of the model itself. According to observations for application developed with the workbenches (DB)² and ID² this third step starts about half a year after introduction of the solution to practice.

Open problem 5.

Develop a theory of schema evolution that allows to change the scope of the model, i.e. the partial reality, and that allows to reason on the impact of these changes.

According to [14], a theory consists of (1) an abstract calculus and a set of postulates and (2) a set of rules that assign an empirical content to the calculus and the postulates by providing ‘coordinating definitions’ and ‘empirical interpretations’. Conceptual modelling typically results in the first part and assumes that the second part is intuitively understood by the user of the model.

extracted three main properties of model relationships $R(\mathcal{P}, \mathcal{R}, \mathcal{A})$ between phenomena, representations and actors developing the model:

**Mapping property:** The modelling relationship is based on a mapping from $\mathcal{P}$ to $\mathcal{R}$ that is a surjective function.

**Truncation property:** The mapping is based on an explicit abstraction, i.e. does not distinguish all possible properties of phenomena but only those that serve the purpose of the mapping. The properties considered (surplus properties) and the ignored ones are clearly separated.

**Pragmatic property:** The modelling relationship is based on the scope of the actors, their intentions and tasks, their culture, skills and knowledge, the purposes of modelling and is based on admissible tools and techniques of investigation.

Open problem 6.

Develop a theory of conceptual modelling that allows to reason on the mapping, truncation and pragmatic properties, and their change. This theory must be based on explicit criteria for choosing a certain schema.

2.3 Modelling By Suites of Models

It is well-known [16] that each schema has a large variety of equivalent schemata. Whether we choose the current one depends

- on the designer knowledge, habits, recency, vividness, and congruity,
- on the underlying technology and the corresponding requirements to avoid certain schemata or to prefer certain schemata, and
- on the application and functional requirements especially performance requirements.

All these reasons for preferring one schema over all others may change. If a change occurs we may switch to another schema or we evolve the current schema to a more appropriate one. The current schema is thus a snapshot within a schema suite. Some of the schemata are equivalent to other, are refinements or extensions of other schemata.
A schema suite consists of a set of schemata, an integration or association schema among these schemata and obligations requiring maintenance of the association. We specify schemata based on a type system enabling in describing structuring of schemata and functionality of schema changes, in describing their associations through relationship types and constraints. The functionality of schemata changes is specified by a retrieval expression, the maintenance policy and a set of functions supporting the utilization of the schema and the schema suite.

Open problem 7.
Develop a theory of isomorphisms, refinement and extension of conceptual schemata.

Databases are the kernel of information systems. We use databases for extraction of data, for creation of data that have a meaning for the application (content), for extraction of data that can be understood by a user (data-warehouse content), for enhancement of data with other data characterising the data (topic-enhanced data), for protection of data against misuse (protected data), for relating data to certain users (private data), for structuring data in a way that can easily be memorised (meme data), etc. Often the generation of such data can be layered in a form similar to the one in Figure 2.

| Layer 4: Memes of the users |
| Layer 3-4: Privacy protection layer |
| Layer 3: Topics of topic landscapes for annotation/representation |
| Layer 2: Concepts of concept bases for foundation/explanation |
| Layer 1: Content of content bases as macro-data or aggregations |
| Layer 0: Data and documents of underlying databases as micro-data |

Fig. 2. An Architecture of Content Management User-Oriented Systems

Each extraction, transformation and loading of data to another database changes also semantics of data, applicability of functions, and pragmatics of data. This change of semantics is well-known for views. Constraints valid in the database used for extraction must not remain to be valid. Other constraints may become valid due to the extraction. Schemata used for loading often use implicit constraints. The same observation can be made for changes in pragmatics, i.e. in the meaning of data for users. Additionally, the operations used for extraction may lead to different meaning or are creating non-sense. This observation can already been made for data warehouses and OLAP databases. A similar change is imposed whenever a database becomes changed due to crucial or inevitable surprises.

Open problem 8.
Develop architectures of layered information systems with appropriate maintenance mechanisms for data at all layers and with transformation of semantics and pragmatics together with extraction and transformation of data.
3 Specification of Structuring

3.1 Languages for Structure Specification

Structuring of databases is based on three interleaved and dependent parts:

**Syntactics:** Inductive specification of structures uses a set of base types, a collection of constructors and an theory of construction limiting the application of constructors by rules or by formulas in deontic logics. In most cases, the theory may be dismissed. **Structural recursion** is the main specification vehicle.

**Semantics:** Specification of admissible databases on the basis of static integrity constraints describes those database states which are considered to be legal. If structural recursion is used then a variant of hierarchical first-order predicate logics may be used for description of integrity constraints.

**Pragmatics:** Description of context and intension is based either on explicit reference to the enterprise model, to enterprise tasks, to enterprise policy, and environments or on intensional logics used for relating the interpretation and meaning to users depending on time, location, and common sense.

The inductive specification of structuring is based on **base types** and **type constructors**. A **base type** is an algebraic structure $B = (\text{Dom}(B), \text{Op}(B), \text{Pred}(B))$ with a name, a set of values in a domain, a set of operations and a set of predicates. A class $B^C$ on the base type is a collection of elements form $\text{dom}(B)$. Usually, $B^C$ is required to be set. It can be a list, a multi-set, a tree etc. Classes may be changed by applying operations. Elements of a class may be classified by the predicates.

A **type constructor** is a function from types to a new type. The constructor can be supplemented with a selector for retrieval (such as Select) and update functions (such as Insert, Delete, and Update) for value mapping from the new type to the component types or to the new type, with correctness criteria and rules for validation, with default rules, with one or more user representations, and with a physical representation or properties of the physical representation.

Typical constructors used for database definition are the set, tuple, list and multi-set constructors. For instance, the set type is based on another type and uses algebra of operations such as union, intersection and complement. The retrieval function can be viewed in a straightforward manner as having a predicate parameter. The update functions such as Insert, Delete are defined as expressions of the set algebra. The user representation is using the braces $\{, \}$. The type constructors define type systems on basic data schemes, i.e. a collection of constructed data sets. In some database models, the type constructors are based on pointer semantics.

Other useful modelling constructs are **naming** and referencing. Each concept type and each concept class has a name. These names can be used for the definition of further types or referenced within a type definition. Often structures include also **optional** components. Optional components and references must be used with highest care since otherwise truly hyper-sophisticated logics such as topoi are required [10]. A better approach to database modelling is the requirement of weak value-identifiability of all database objects [11].
3.2 Integrity Constraints

Integrity constraints are used to separate “good” states or sequences of states of a database system from those which are not intended. They are used for specification of semantics of both structures and processes. Therefore, consistency of database applications can not be treated without constraints. At the same time, constraints are given by users at various levels of abstraction, with a variety of vagueness and intensions behind and on the basis of different languages. For treatment and practical use, however, constraints must be specified in a clear and unequivocal form and language. In this case, we may translate these constraints to internal system procedures which are supporting consistency enforcement.

Each structure is also based on a set of implicit model-inherent integrity constraints:

Component-construction constraints are based on existence, cardinality and inclusion of components. These constraints must be considered in the translation and implication process.

Identification constraints are implicitly used for the set constructor. Each object either does not belong to a set or belongs only once to the set. Sets are based on simple generic functions. The identification property may be, however, only representable through automorphism groups. We shall later see that value-representability or weak-value representability lead to controllable structuring.

Acyclicity and finiteness of structuring supports axiomatisation and definition of the algebra. It must, however, be explicitly specified. Constraints such as cardinality constraints may be based on potential infinite cycles.

Superficial structuring leads to representation of constraints through structures. In this case, implication of constraints is difficult to characterize.

Implicit model-inherent constraints belong to the performance and maintenance traps.

Integrity constraints can be specified based on the B(eri-)V(ardi)-frame, i.e. by an implication with a formula for premises and a formula for the implication. BV-constraints do not lead to rigid limitation of expressibility. If structuring is hierarchic then BV-constraints can be specified within the first-order predicate logic. We may introduce a variety of different classes of integrity constraints defined:

Equality-generating constraints allow to generate for a set of objects from one class or from several classes equalities among these objects or components of these objects.

Object-generating constraints require the existence of another object set for a set of objects satisfying the premises.

A class \( C \) of integrity constraints is called Hilbert-implication-closed if it can be axiomatised by a finite set of bounded derivation rules and a finite set of axioms. It is well-known that the set of join dependencies is not Hilbert-implication-closed for relational structuring. However, an axiomatisation exists with an unbounded rule, i.e. a rule with potentially infinite premises.
The main deficiency is the constraint acquisition problem. Since we need a treatment for sets a more sophisticated reasoning theory is required. One good candidate is visual or graphical reasoning that goes far beyond logical reasoning [3].

Open problem 9.

Provide a reasoning facility for treatment of sets of constraints.

Classify 'real life' constraint sets which can be easily maintained and specified.

Additional problems for dependencies can already be stated on the level of the relational model. We conclude this subsection with a number of problems.

Open problem 10.

Is the implication problem for closure dependencies and functional dependencies decidable? Axiomatizable?

Which subclass of inclusion constraints properly containing the unary inclusion dependencies is axiomatizable together with the class of functional dependencies?

Which subclass of join dependencies properly containing the class of multi-valued dependencies is axiomatizable?

Characterize relations which are compatible under functional dependencies.

Characterize the properties of constraint classes under horizontal decomposition.

3.3 Representation Alternatives

The classical approach to database objects is to store an object based on strong typing. Each real life thing is thus represented by a number of objects which are either coupled by the object identifier or supported by specific maintenance procedures. In general, however, we might consider two different approaches to representation of objects:

Class-wise, identification-based representation: Things of reality may be represented by several objects. The object identifier (OID) supports identification without representing the complex real-life identification. Objects can be elements of several classes. In the early days of object-orientation it has been assumed that objects belong to one and only one class. This assumption has led to a number of migration problems which have not got any satisfying solution.

Structuring based on extended ER models [16] or object-oriented database systems uses this option. Technology of relational and object-relational database systems is based on this representation alternative.

Object-wise representation: Graph-based models which have been developed in order to simplify the object-oriented approaches [1] display objects by their sub-graphs, i.e. by the set of nodes associated to a certain object and the corresponding edges. This representation corresponds to the representation used in standardization.

XML is based on object-wise representation. It allows to use null values without notification. If a value for an object does not exist, is not known, is not applicable or cannot be obtained etc. the XML schema does not use the tag corresponding to the attribute or the component. Classes are hidden.
Object-wise representation has a high redundancy which must be maintained by the system thus decreasing performance to a significant extent. Beside the performance problems such systems also suffer from low scalability and bad utilization of resources. The operating of such systems leads to lock avalanches. Any modification of data requires a recursive lock of related objects.

For these reasons, objects-wise representation is applicable only under a number of restrictions:

- The application is stable and the data structures and the supporting basic functions necessary for the application are not changed during the lifespan of the system.
- The data set is almost free of updates. Updates, insertions and deletions of data are only allowed in well-defined restricted ‘zones’ of the database.

A typical application area for object-wise storage are archiving systems, information presentation systems, and content management systems. They use an update system underneath. We call such systems play-out system. The data are stored in the way in which they are transferred to the user. The data modification system has a play-out generator that materializes all views necessary for the play-out system.

Other applications are main-memory databases without update. The SAP database system uses a huge set of related views.

We may use the first representation for our storage engine and the second representation for the input engine or the output engine in data warehouse approaches.

**Open problem 11.**  
Find techniques and theories that support treatment of redundant sets of objects and that support consistency management for sets of objects, e.g., XML document sets.

Database optimization is based on the knowledge of complexity of operations. If we know that a certain set of operations is far more complex than the rest and if we know a number of equivalent representations then we can choose among those the less complex one. A typical case of optimization is the vertical normalization where a relation is decomposed into a set of relations which has less representation complexity and which is simpler to support. Horizontal normalization selects parts of relations with lower complexity. Deductive normalization reduces relations to those elements than cannot be generated from the other elements by generation rules. At the same time we require that the representations are equivalent. So far, these three kinds of normalization are treated in a separate form.

**Open problem 12.**  
Find a common framework for the utilization of vertical, horizontal and deductive normalization for object-relational database models.

Normalization is often based on database constraints. In order to get a correct normalization we need to know the entire set of valid constraints in the given application. This is infeasible and often not achievable.

**Open problem 13.**  
Find a normalization theory which is robust for incomplete constraint sets.
4 Specification of Functionality

4.1 Operations for Information Systems

General operations on type systems can be defined by structural recursion. Given types $T$, $T'$ and a collection type $C_T$ on $T$ (e.g. set of values of type $T$, bags, lists) and operations such as generalized union $\cup C_T$, generalized intersection $\cap C_T$, and generalized empty elements $\emptyset C_T$ on $C_T$. Given further an element $h_0$ on $T'$ and two functions defined on the types $h_1 : T \rightarrow T'$ and $h_2 : T' \times T' \rightarrow T'$.

Then we define the structural recursion by insert presentation for $R C_T$ on $T$ as follows

\[
srec h_0, h_1, h_2 (\emptyset C_T) = h_0
\]

\[
srec h_0, h_1, h_2 ([|s|]) = h_1(s) \quad \text{for singleton collections } [|s|]
\]

\[
srec h_0, h_1, h_2 ([|s|] \cup C_T R C_T) = h_2(h_1(s), s\text{ rec } h_0, h_1, h_2 (R C_T)) \quad \text{iff } [|s|] \cap C_T R C_T = \emptyset C_T.
\]

All operations of the object-relational database model, the extended entity-relationship model and of other declarative database models can be defined by structural recursion, e.g.,

- selection is defined by $srec_{\emptyset, \iota, \cup}$ for the function

  \[
  \iota_\alpha([a]) = \begin{cases} \{a\} &\text{if } \{a\} \models \alpha \\ \emptyset &\text{otherwise} \end{cases}
  \]

- aggregation functions can be defined based on the two functions for null values

  \[
  h^0_f(s) = \begin{cases} 0 &\text{if } s = \text{NULL} \\ f(s) &\text{if } s \neq \text{NULL} \end{cases}
  \]

  \[
  h^{undef}_f(s) = \begin{cases} \text{undef} &\text{if } s = \text{NULL} \\ f(s) &\text{if } s \neq \text{NULL} \end{cases}
  \]

  through structural recursion, e.g.,

  \[
  \text{sum}_{null}^0 = srec_{0, h^0_f, +} \quad \text{or } \text{sum}_{null}^{undef} = srec_{0, h^{undef}_f, +} ;
  \]

  \[
  \text{count}_{null}^1 = srec_{0, h^0_f, +} \quad \text{or } \text{count}_{null}^{undef} = srec_{0, h^{undef}_f, +}
  \]

  or the doubtful SQL definition of the average function

  \[
  \frac{\text{sum}_{null}^0}{\text{count}_{null}^1}.
  \]

Similarly we may define intersection, union, difference, projection, join, nesting and unnesting, renaming, insertion, deletion, and update.

Structural recursion is also limited in expressive power. Nondeterministic while tuple-generating programs (or object generating programs) cannot be expressed.

Operations may be either used for retrieval of values from the database or for state changes within the database.

The general frame for operation definition in the co-design approach is based on views used to restrict the scope, pre-, and postconditions used to restrict the applicability and the activation of operations and the explicit description of enforced operations:
Operation $\varphi$

[View: $<$ View_Name $>$]
[Precondition: $<$ Activation.Condition $>$]
[ActivatedOperation: $<$ Specification $>$]
[Postcondition: $<$ Acceptance.Condition $>$]
[EnforcedOperation: $<$ Operation, Condition $>$]

The relational model and object-relational models are extended by aggregation, grouping and bounded recursion operations. The semantics of these operations varies among database management systems and has not found yet a mathematical basis [6].

Open problem 14.
Develop a general theory of extended operations for object-relational models.

4.2 Dynamic Integrity Constraints

Database dynamics is defined on the basis of transition systems. A transition system on the schema $S$ is a pair

$$TS = (S, \{ \overset{\alpha}{\rightarrow} | \alpha \in \mathcal{L} \})$$

where $S$ is a non-empty set of state variables,
$\mathcal{L}$ is a non-empty set (of labels),
and $\overset{\alpha}{\rightarrow} \subseteq S \times (S \cup \{\infty\})$ for each $\alpha \in \mathcal{L}$.

State variables are interpreted by states. Transitions are interpreted by transactions on $S$.

Database lifetime is specified on the basis of paths on $TS$. A path $\pi$ through a transition system is a finite or $\omega$ length sequence of the form $s_0 \overset{\alpha_1}{\rightarrow} s_1 \overset{\alpha_2}{\rightarrow} ...$. The length of a path is its number of transitions.

For the transition system $TS$ we can introduce now a temporal dynamic database logic using the quantifiers $\forall_f$ (always in the future), $\forall_p$ (always in the past), $\exists_f$ (sometimes in the future), $\exists_p$ (sometimes in the past).

First-order predicate logic can be extended on the basis of temporal operators. The validity function $I$ is extended by time. Assume a temporal class $(RC, l_R)$. The validity function $I$ is extended by time and is defined on $S(ts, RC, l_R)$. A formula $\alpha$ is valid for $I_{RC, l_R}(\alpha, ts)$ if it is valid on the snapshot defined on $ts$, i.e., $I_{RC, l_R}(\alpha, ts) = 1$ iff $I_{S(ts, RC, l_R)}(\alpha, ts)$.

- For formulas without temporal prefix the extended validity function coincides with the usual validity function.
- $I(\forall_f \alpha, ts) = 1$ iff $I(\alpha, ts') = 1$ for all $ts' > ts$;
- $I(\forall_p \alpha, ts) = 1$ iff $I(\alpha, ts') = 1$ for all $ts' < ts$;
- $I(\exists_f \alpha, ts) = 1$ iff $I(\alpha, ts') = 1$ for some $ts' > ts$;
- $I(\exists_p \alpha, ts) = 1$ iff $I(\alpha, ts') = 1$ for some $ts' < ts$. 
The modal operators $\forall p$ and $\exists p$ ($\forall f$ and $\exists f$ respectively) are dual operators, i.e. the two formulas $\forall h \alpha$ and $\neg \exists h \neg \alpha$ are equivalent. These operators can be mapped onto classical modal logic with the following definition:

$$\square \alpha \equiv (\forall f \alpha \land \forall p \alpha \land \alpha);$$

$$\Diamond \alpha \equiv (\exists f \alpha \lor \exists p \alpha \lor \alpha).$$

In addition, temporal operators until and next can be introduced.

The most important class of dynamic integrity constraint are state-transition constraints $\alpha O \beta$ which use a pre-condition $\alpha$ and a post-condition $\beta$ for each operation $O$. The state-transition constraint $\alpha O \beta$ can be expressed by the the temporal formula $\alpha \xrightarrow{O} \beta$.

Each finite set of static integrity constraints can be equivalently expressed by a set of state-transition constraints $\{ \land_{\alpha \in \Sigma} \alpha \xrightarrow{O} \land_{\alpha \in \Sigma} \alpha \mid O \in Alg(M) \}$.

Integrity constraints may be enforced

- either at the procedural level by application of
  - trigger constructs [8] in the so-called active event-condition-action setting,
  - greatest consistent specialisations of operations [10],
  - or stored procedures, i.e., fully fledged programs considering all possible violations of integrity constraints,
- or at the transaction level by restricting sequences of state changes to those which do not violate integrity constraints,
- or by the DBMS on the basis of declarative specifications depending on the facilities of the DBMS,
- or at a the interface level on the basis of consistent state changing operations.

Database constraints are classically mapped to transition constraints. These transition constraints are well-understood as long as they can be treated locally. Constraints can thus be supported using triggers or stored procedures. Their global interdependence is, however, an open issue.

Open problem 15.

Develop a theory of interference of database constraints that can be mapped to well-behaving sets of database triggers and stored procedures.

### 4.3 Specification of Workflows

A large variety of approaches to workflow specification has been proposed in the literature. We prefer formal descriptions with graphical representations and thus avoid pitfalls of methods that are entirely based on graphical specification such as the and/or traps. We use basic computation step algebra introduced in [20]:

- Basic control commands are sequence $;$ (execution of steps in sequence), parallel split $\mid$ (execute steps in parallel), exclusive choice $\|$ (choose one execution path from many alternatives), synchronization $\|_{\text{sync}}$ (synchronize two parallel threads of execution by an synchronization condition $\text{sync}$, and simple merge $+$ (merge two alternative execution paths). The exclusive choice is considered to be the default parallel operation and is denoted by $\|$. 
• **Structural control commands** are arbitrary cycles * (execute steps w/out any structural restriction on loops), arbitrary cycles + (execute steps w/out any structural restriction on loops but at least once), optional execution [] (execute the step zero times or once), implicit termination \ (terminate if there is nothing to be done), entry step in the step \ and termination step in the step \.

The basic computation step algebra may be extended by **advanced step commands**:

• **Advanced branching and synchronization control commands** are multiple choice \( [m,n] \) (choose between \( m \) and \( n \) execution paths from several alternatives), multiple merge (merge many execution paths without synchronizing), discriminator (merge many execution paths without synchronizing, execute the subsequent steps only once) n-out-of-m join (merge many execution paths, perform partial synchronization and execute subsequent step only once), and synchronizing join (merge many execution paths, synchronize if many paths are taken, simple merge if only one execution path is taken).

• We also may define **control commands on multiple objects (CMO)** such as CMO with a priori known design time knowledge (generate many instances of one step when a number of instances is known at the design time), CMO with a priori known runtime knowledge (generate many instances of one step when a number of instances can be determined at some point during the runtime (as in FOR loops)), CMO with no a priori runtime knowledge (generate many instances of one step when a number of instances cannot be determined (as in a while loop)), and CMO requiring synchronization (synchronization edges) (generate many instances of one activity and synchronize afterwards).

• **State-based control commands** are deferred choice (execute one of the two alternative threads, the choice which thread is to be executed should be implicit), interleaved parallel executing (execute two activities in random order, but not in parallel), and milestone (enable an activity until a milestone has been reached).

• Finally, **cancellation control commands** are used, e.g. cancel step (cancel (disable) an enabled step) and cancel case (cancel (disable) the case).

These control composition operators are generalizations of workflow patterns and follow approaches developed for Petri net algebras.

Operations defined on the basis of this general frame can be directly translated to database programs. So far no theory of database behavior has been developed that can be used to explain the entire behavior and that explain the behavior in depth for a run of the database system. A starting point for the development of such theory might be the proposal [17] to use abstract state machines [2].

**Open problem 16.**

Develop a theory of database behavior. This theory explains the run of database management systems as well as the run of database systems.

### 4.4 Architecture of Database Engines

Operating of information systems is modelled by separating the systems state into four state spaces:
\( \mathcal{E} \mathcal{R} \mathcal{C} = (\text{input states} \ \mathcal{I} \mathcal{N}, \ \text{output states} \ \mathcal{O} \mathcal{U} \mathcal{T}, \ \text{engine states} \ \mathcal{D} \mathcal{B} \mathcal{M} \mathcal{S}, \ \text{database states} \ \mathcal{D} \mathcal{B}) \).

The input states accommodate the input to the database system, i.e. queries and data. The output space allows to model the output of the DBMS, i.e. output data of the engine and error messages. The internal state space of the engine is represented by engine states. The database content of the database system is represented in the database states. The four state spaces can be structured. This structuring is reflected in all four state spaces. For instance, if the database states are structured by a database schema then the input states are accordingly structured.

Using value-based or object-relational models the database states can be represented by relations. An update imposed to a type of the schema is in this case a change to one of the relations. State changes are modeled on the basis of abstract state machines \[2\] through state change rules. An engine is specified by its programs and its control. We follow this approach and distinguish between programs that specify units of work or services and meet service quality obligations and control and coordination that is specified on the level of program blocks with or without atomicity and consistency requirements or specified through job control commands.

Programs are called with instantiated parameters for their variables. Variables are either static or stack or explicit or implicit variables. We may use furthermore call parameters such as \texttt{onSubmit} and \texttt{presentationMode}, priority parameters such as \texttt{onFocus} and \texttt{emphasisMode}, control parameters such as \texttt{onRecovery} and \texttt{hookOnProcess}, error parameter such as \texttt{onError} and \texttt{notifyMode}, and finally general transfer parameters such as \texttt{onReceive} and \texttt{validUntil}.

Atomicity and consistency requirements are supported by the variety of transaction models. Typical examples are flat transactions, sagas, join-and-split transactions, contracts or long running activities \[16\].

State changes \(T(s_1, ..., s_n) := t\) of a sub-type \(T'\) of the database engine \(\mathcal{E} \mathcal{R} \mathcal{C}\). A set \(\mathcal{U} = \{T_i(s_{i,1}, ..., s_{i,n_i}) := o_i | 1 \leq i \leq m\}\) of object-based state changes is consistent, if the equality \(o_i = o_j\) is implied by \(T_i(s_{i,1}, ..., s_{i,n_i}) = T_j(s_{j,1}, ..., s_{j,n_j})\) for \(1 \leq i < j \leq m\).

The result of an execution of a consistent set \(\mathcal{U}\) of state changes leads to a new state \(\mathcal{E} \mathcal{R} \mathcal{C}\) to \(\mathcal{E} \mathcal{R} \mathcal{C} + \mathcal{U}\)

\[
(\mathcal{E} \mathcal{R} \mathcal{C} + \mathcal{U})(o) = \begin{cases} 
  \text{Update}(T_i, s_{i,1}, ..., s_{i,n_i}, o_i) \\
  \text{if } T_i(s_{i,1}, ..., s_{i,n_i}) := o_i \in \mathcal{U} \\
  \mathcal{E} \mathcal{R} \mathcal{C}(o) \text{ in the other case}
\end{cases}
\]

for objects \(o\) of \(\mathcal{E} \mathcal{R} \mathcal{C}\).

A parameterized program \(r(x_1, ..., x_n) = P\) of arity \(n\) consists of a program name \(r\), a transition rule \(P\) and a set \(\{x_1, ..., x_n\}\) of free variables of \(P\).

An information system \(\mathcal{E} \mathcal{R} \mathcal{C}\) is a model of \(\phi\) (\(\mathcal{E} \mathcal{R} \mathcal{C} \models \phi\)) if \(\phi\)\(\mathcal{E} \mathcal{R} \mathcal{C} = \text{true}\) for all variable assignments \(\zeta\) for free variables of \(\phi\).
Two typical program constructors are the execution of a program for all values that satisfy a certain restriction

\[
\text{FOR ALL } x \text{ WITH } \phi \rightarrow \text{DO } P
\]

and the repetition of a program step in a loop

\[
\text{DO } P_1
\]

\[
\text{LOOP } \alpha
\]

We introduce also other program constructors such as sequential execution, branch, parallel execution, execution after value assignment, execution after choosing an arbitrary value, skip, modification of a information system state, and call of a subprogram.

We use the abstract state machine approach also for definition of semantics of the programs.

A transition rule \( P \) leads to a set \( \mathcal{U} \) of state changing operations in a state \( \mathcal{E} \mathcal{R} \mathcal{C} \) if it is consistent. The state of the information system is changed for a variable assignment \( \zeta \) to \( \text{yields}(P, \mathcal{E} \mathcal{R} \mathcal{C}, \zeta, \mathcal{U}) \).

Semantics of transition rules is defined in a calculus that uses rules of the form

\[
\text{prerequisite}_1, \ldots, \text{prerequisite}_n \quad \text{conclusion} \quad \text{where condition}
\]

For instance, the state change imposed by the first program step is defined by

\[
\forall a \in I : \text{yields}(P, \mathcal{E} \mathcal{R} \mathcal{C}, \zeta[x \mapsto a], \mathcal{U}_a)
\]

\[
\text{yields}(\text{FOR ALL } x \text{ WITH } \phi \text{ DO } P, \mathcal{E} \mathcal{R} \mathcal{C}, \zeta, \bigcup_{a \in I} \mathcal{U}_a)
\]

where \( I = \text{range}(x, \phi, \mathcal{E} \mathcal{R} \mathcal{C}, \zeta) \)

The range \( \text{range}(x, \phi, \mathcal{E} \mathcal{R} \mathcal{C}, \zeta) \) is defined by the set \( \{ o \in \mathcal{E} \mathcal{R} \mathcal{C} | \llbracket \phi \rrbracket_{(x \mapsto a)} = \text{true} \} \).

Open problem 17.

Develop a general theory of abstraction and refinement for database systems that supports architectures and separation or parquetting of database systems into components.

4.5 Data Extraction Frameworks

Surprises, data warehousing, and complex applications often require sophisticated data analysis. The most common approach to data analysis is to use data mining software or reasoning systems based on artificial intelligence. These applications allow to analyse data based on the data on hand. At the same time data are often observational or sequenced data, noisy data, null-valued data, incomplete data, of wrong granularity, of wrong precision, of inappropriate type or coding, etc. Therefore, brute-force application of analysis algorithms leads to wrong results, to loss of semantics, to misunderstandings etc.
We thus need general frameworks for data analysis beyond the framework used for
data mining. We may use approaches known from mathematics for the development of
data analysis framework.

**Open problem 18.**

Develop semantic-preserving and pragmatic-preserving frameworks for data
extraction.

5 Specification of Interactivity

Interactivity of information systems has been mainly considered on the level of presenta-
tion systems by the Arch or Seeheim separation between the application system and
the presentation system. Structuring and functionality are specified within a database
modelling language and its corresponding algebra. Pragmatics is usually not consid-
ered within the database model. The interaction with the application system is based on
a set of views which are defined on the database structure and are supported by some
functionality.

![Fig. 3. Architecture of Information Systems Enhanced by Presentation Systems Compared With the Architecture of Web Information Systems](image)

The general architecture of a web information system is shown in Figure 3. This
architecture has successfully been applied in more than 30 projects resulting in huge
or very large information-intensive websites and in more than 100 projects aiming in
building large information systems.

In the co-design framework we generalize this approach by

**introduction of media objects** which are generalized views that have been extended
by functionality necessary, are adapted to the users needs and delivered to the actor
by a container [12] and by
introduction of story spaces [13] which specify the stories of usage by groups of users (called actors) in their context, can be specialized to the actual scenario of usage and use a variety of play-out facilities.

User interaction modelling involves several partners (grouped according to characteristics; group representatives are called ‘actors’), manifests itself in diverse activities and creates an interplay between these activities. Interaction modelling includes modelling of environments, tasks and actors beside modelling of interaction flow, interaction content and interaction form.

5.1 Story Space

Modelling of interaction must support multiple scenarios. In this case, user profiles, user portfolios, and the user environment must be taken into consideration. The story of interaction is the intrigue or plot of a narrative work or an account of events. The language SiteLang [20] offers concepts and notation for specification of story spaces, scene and scenarios in them.

Within a story one can distinguish threads of activity, so-called scenarios, i.e., paths of scenes that are connected by transitions. We define the story space \( \Sigma_W \) as the 7-tuple \((S_W, T_W, E_W, G_W, A_W, \lambda_W, \kappa_W)\) where \( S_W \), \( T_W \), \( E_W \), \( G_W \) and \( A_W \) are the set of scenes created by \( W \), the set of scene transitions and events that can occur, the set of guards and the set of actions that are relevant for \( W \), respectively. Thus, \( T_W \) is a subset of \( S_W \times S_W \). Furthermore \( \lambda_W : S_W \rightarrow \text{SceneSpec} \) is a function associating a scene specification with each scene in \( S_W \), and \( \kappa_W : T_W \rightarrow E_W \times G_W \times A_W \), \( t \mapsto (e, g, a) \) is a function associating with each scene transition \( t \) occurring in \( W \) the event \( e \) that triggers transition \( t \), the guard \( g \), i.e. a logical condition blocking the transition if it evaluates to false on occurrence of \( e \), and the action \( a \) that is performed while the transition takes place.

We consider scenes as the conceptual locations at which the interaction, i.e., dialogue takes place. Dialogues can be specified using so-called dialogue-step expressions. Scenes can be distinguished from each other by means of their identifier: Scene-ID. With each scene there is associated a media object and the set of actors that are involved in it. Furthermore, with each scene a representation specification is associated as well as a context. Scenes therefore can be specified using the following frame:

\[
\text{Scene} = (\text{Scene-ID} \\
\text{DialogueStepExpression} \\
\text{Data views with associated functions} \\
\text{User} \\
\quad \text{UserID} \\
\quad \text{UserRight} \\
\quad \text{UserTasksAssigned} \\
\quad \text{UserRoles} \\
\text{Representation (styles, defaults, emphasis, ...)} \\
\text{Context (equipment, channel, particular)}
\]

Dialogue-step expressions consist of dialogues and operators applied to them. A typical scene is displayed in Figure 4. A learner may submit solutions in the data mining
cup. Before doing so, the user must pay a certain fee if she has not already been paying. The system either knows the user and his/her profile. If the user has already paid the fee then the payment dialogue step is not shown. If the user has not paid the fee or is an anonymous user then the fee dialogue step must be visited and the dialogue step for task completion is achievable only after payment.

5.2 Media Type Suite

Media types have been introduced in [12]. Since users have very different needs in data depending on their work history, their portfolio, their profile and their environment we send the data packed into containers. Containers have the full functionality of the view suite. Media type suites are based on view suites and use a special delivery and extraction facility. The media type suite is managed by a system consisting of three components:

Media object extraction system: Media objects are extracted and purged from database, information or knowledge base systems and summarized and compiled into media objects. Media objects have a structuring and a functionality which allows to use these in a variety of ways depending on the current task.

Media object storage and retrieval system: Media objects can be generated on the fly whenever we need the content or can be stored in the storage and retrieval subsystem. Since their generation is usually complex and a variety of versions must be kept, we store these media objects in the subsystem.

Media object delivery system: Media objects are used in a large variety of tasks, by a large variety of users in various social and organizational contexts and further in various environments. We use a media object delivery system for delivering data to the user in form the user has requested. Containers contain and manage the set of media object that are delivered to one user. The user receives the user-adapted container and may use this container as the desktop database.
This understanding closely follows the data warehouse paradigm. It is also based on the classical model-view-control paradigm. We generalize this paradigm to media objects, which may be viewed in a large variety of ways and which can be generated and controlled by generators.

Open problem 19.

Provide a theory that support adaptable (to the user, to the context, to the history, to the environment) delivery of content and reception of content.

6 Integrating Specification Aspects into Co-design

The languages introduced so far seem to be rather complex and the consistent development of all aspects of information systems seems to be rather difficult. We developed a number of methodologies to development in order to overcome difficulties in consistent and complete development. Most of them are based on top-down or refinement approaches that separate aspects of concern into abstraction layers and that use extension, detailisation, restructuring as refinement operations.

6.1 The Abstraction Layer Model for Information Systems Development

We observe that information systems are specified at different abstraction layers:

1. The strategic layer addresses the purpose of the information system, i.e. its mission statement and the anticipated customer types including their goals. The results of the design process are conducted into the stakeholder contract specification.

2. The requirements elaboration layer describes the information system, analyzes business processes and aims in elicitation of the requirements to the information system. The results of the design process are combined into the system specification.

3. The business layer deals with modelling the anticipated usage of the information system in terms of customer types, locations of the information space, transitions between them, and dialogues and discourses between categories of users (called actors). The result of this abstraction layer is compiled into an extended system manual including mockups of the interfaces and scenarios of utilization.

4. The conceptual layer integrates the conceptual specification of structuring, functionality, distribution and interactivity. The results of this step are the database schema, the workflows, the view and media type suites, the specification of distribution, and the story space.

5. At the implementation layer, logical and physical database structures, integrity enforcement procedures, programs, and interfaces are specified within the language framework of the intended platform. The result of specification at the implementation layer is the implementation model. This model is influenced by the builder of the information system.

6. The exploitation layer is not considered here. Maintenance, education, introduction and administration are usually out of the scope of conceptualization of an application.
6.2 The Co-design Methodology

Methodologies must confirm both with the SPICE v. 2.0 and SW-CMM v. 2.0 requirements for consistent system development. The co-design framework is based on a step-wise refinement along the abstraction layers. Since the four aspects of information systems - structuring, functionality, distribution and interactivity - are interrelated they cannot be developed separately. The methodology sketched below is based on steps in the following specification frame.

The steps used in one of the methodologies are:

**Strategic layer**
1. Developing visions, aims and goals
2. Analysis of challenges and competitors

**Requirements elaboration layer**
3. Separation into system components
4. Sketching the story space
5. Sketching the view suite
6. Specifying business processes

**Business user layer**
7. Development of scenarios of the story space
8. Elicitation of main data types and their associations
9. Development of kernel integrity constraints, e.g., identification constraints
10. Specification of user actions, usability requirements, and sketching media types
11. Elicitation of ubiquity and security requirements
<table>
<thead>
<tr>
<th>Rule # i</th>
<th>Name of the step</th>
<th>Task 1.</th>
<th>Task 2.</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Used documents**
- Documents of previous steps
- (IS development documents)
- Customer documents and information

**Documents under change**
- IS development documents
- Contracts

**Aims, purpose and subject**
- General aims of the step
- Agreed goals for this step
- Purpose
- Matter, artifact

**Actors involved**
- Actor A, e.g., customer representatives
- Actor B, e.g., developer

**Theoretical foundations**
- Database theory
- Organization theory
- Computer science
- Cognition, psychology, pedagogic

**Methods and heuristics**
- Syntax and pragmatics
- Used specification languages
- Simplification approaches

**Developed documents**
- IS development documents
- Results and deliverables

**Results**
- Information gathering conditions
- Fulfilled by the customer side
- Information dependence conditions
- Conditions on the participation

**Enabling condition for step**
- Completeness and correctness criteria
- Sign-offs, contracts
- Quality criteria
- Obligation for the step fulfilled

---

**Conceptual layer**

12. Specification of the story space
13. Development of data types, integrity constraint, their enforcement
14. Specification of the view suite and distribution
15. Development of workflows
16. Control of results by sample data, sample processes, and sample scenarios
17. Specification of the media type suite
18. Modular refinement of types, views, operations, services, and scenes
19. Normalization of structures
20. Integration of components along architecture

**Implementation layer**

21. Transformation of conceptual schemata into logical schemata, programs, and interfaces
22. Development of distribution
23. Developing solutions for performance improvement, tuning
24. Transformation of logical schemata into physical schemata
25. Checking durability, robustness, scalability, and extensibility

The co-design methodology has been practically applied in a large number of information system projects and has nevertheless a sound theoretical basis. We do not want to compete with UML but support system development at a sound basis without ambiguity, ellipses and conceptual mismatches.

Open problem 20.
Develop quality characteristics and measurements for each of the modelling steps and measures for quality preserving transformations.

7 Conclusion

Database and information systems research has brought up a technology that is now a part of the everyday infrastructure. Database systems are running as embedded systems, e.g. for car information systems, as collaborating systems or as stand-alone systems. The technology has got a maturity that allows to use and to install a database system whenever information-intensive computation supports an application. Due to this wide range of applications information systems are currently extended to support distributed computation and web information systems. These novel directions of research are currently in intensive change and attract a lot of research. Within this paper we developed a proposal for extending classical database technology to these novel areas. This proposal is only one way of ending and applying the current technology.

At the same time a number of problems remains still open in the area of databases. We have summarized the achievements of a theory of database structuring and functionality. Some of these achievements become already neglected with the development of novel information system models. They are, however, neatly combinable with the novel models. In the first part of the paper we summarized these achievements and introduced some of the main open problems of current database research.

References


Remark: Our main aim has been the survey of current database research. We restrict thus the bibliography only to those references which are necessary for this paper. An extensive bibliography on relevant literature in this field can be found in [15].
Methodik zum Co-Design von Informationssystemen

Bernhard Thalheim

Computer Science Institute, Christian-Albrechts-University Kiel,
Olshausenstrasse 40, 24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de

submitted to the special issue for Erich Ortner organised by Elisabeth Heinemann

1 Herangehensweisen zur Spezifikation von Informationssystemen


Dieser Methodik sind eine Reihe von methodischen und inhaltlichen Brüchen eigen. Die Schwierigkeit eines kompletten Datenbankentwurfs ist jedoch in vielen Fällen auf diese Brüche zurückzuführen. Es werden unterschiedliche Sprachen verwendet und unterschiedliche Personen bringen unterschiedliche Interpretation und Sichtweisen auf das Informationssystem ein.

Für die Spezifikation aller Aspekte von Informationssystemen, d.h. der Strukturierung, Funktionalität, Verteilung und Interaktivität entwickeln wir eine Reihe von miteinander integrierten Spezifikationssprachen. Die Sprachen unterstützen die Entwicklung von Informationssystemen mit Hilfe des Abstraktionsschichtenmodells.


Der Entwurfsprozeß ist ein Prozeß des Abstrahierens und des Konstruierens. Wir können deshalb die unterschiedlichen Abstraktionsarten und Konstruktionsarten miteinander vergleichen.

Mit dem Zachman-Zugang [IZG97] können wir beim Konstruieren unterschiedliche Aspekte von Informationssystemen unterscheiden:

1Ein Ausnahme ist das im Universitätsverbund entwickelte Werkzeug RADD sein [AAB98].
Strukturierung (was): Die Strukturierung der Anwendung wird durch Datenbankmodelle angegeben. Datenbanklehrbücher konzentrieren sich meist auf diesen Aspekt.

Funktionalität (wie): Funktionen und Prozesse, die für die Manipulation und das Retrieval benötigt werden, werden meist erst mit der Entwicklung der Funktionalität der Anwendung auf dem Niveau der Implementierung betrachtet. Da aber die Optimierung des Verhaltens der Anwendung eine dedizierte Unterstützung durch die Strukturierung erfahren muß, sollte die Spezifikation der Funktionalität und der Strukturierung abgestimmt erfolgen.


Motivation (warum): Die Akzeptanz der Systeme wird stark durch die Motivation der Akteure mit bestimmt. Wir verallgemeinern die Motivationsschicht zur allgemeinen Benutzbarkeitsschicht.

Metaaspekte werden im Zachman-Modell bis auf die Motivation nicht betrachtet. Beispiele solcher Kategorien sind Qualitätskategorien wie Allgegenwart, Sicherheit, Konsistenz, Bedeutungstreu, Robustheit, Skalierbarkeit und Dauerhaftigkeit.

Benutzungsaspekte werden im Zachman-Modell vernachlässigt. Es gehören hierzu insbesondere das Aufgabenportfolio und das Organisationsmodell.

Unser Modell der Entwicklung von Informationssystemen im Co-Design-Zugang folgt den ersten drei Aspekten (Strukturierung, Funktionalität und Verteilung) und betrachtet anstelle der letzten drei Aspekte das Storyboard, d.h. die Interaktivität.

Wir fügen dem Zachman-Modell noch weitere Dimensionen hinzu:

Kompetenz (wofür): Es werden die Aufgaben, die durch das Informationssystem unterstützt werden sollen explizit dargestellt.


Qualitätsgarantien (in welcher Qualität): Es wird explizit dargestellt, inwieweit bestimmte Qualitätskriterien durch das System unterstützt werden und welche Qualitätskriterien nicht oder nur bedingt erfüllt werden.
Laufzeitcharakteristiken (wie derzeit): Da die Arbeitsumgebung auch durch Ausnahmesituationen, durch aktuelle Parameter, durch zeitweilige Verschiebung der notwendigen Schritte zum Abchluss und durch benutzungsspezifische Aspekte geprägt ist, sollte die Anpassung des Systems an die Arbeitssituation auch explizit modelliert werden.


Diese Dimensionen untersetzen z.T. die Zachman-Dimensionen. Da im Verlaufe des Modellierungsprozesses alle Aspekte der Anwendung explizit dargestellt werden sollten, umfaßt unsere Methodik auch diese Betrachtungswinkel.

Im Abstraktionsprozeß kann man unterschiedliche Aspekte betrachten. Wir unterscheiden drei Abstraktionsarten:

- Die Komponentenabstraktion kann aufgrund unterschiedlicher Konstruktoren unterschiedliche Ausprägungen besitzen:
  - Die Klassenabstraktion orientiert sich auf die Unterscheidung von Instantierung und Klassifizierung.
  - Die Konstruktorabstraktion orientiert sich an der Benutzung der im Datenbankmodell vorhandenen Konstruktoren. Daraus resultieren Operationen wie die Aggregation und die Dekomposition.
  - Die Beziehungen zwischen Klassen können explizit modelliert sein.
    * Durch Teiltypenhierarchien werden die Generalisierung und Spezialisierung von Klassen dargestellt.
    * Die Konstruktionsbeziehungen folgen meist der Definitionsbeziehung.
    * Abbildungsbeziehungen werden für Datenbanken auf die Sichtenmodellierung reduziert.

- Die Kontextabstraktion ist eine Verallgemeinerung der Lokalisierungsabstraktion und orientiert auf eine Verallgemeinerung ohne Bezug zur konkreten Umgebung.
  - Die Wiederholung von Funktionen kann sowohl für unterschiedliche Strukturen als auch unterschiedliche Teile der Anwendung sinnvoll sein.
  - Die Verteilungssstraktion auf der Grundlage eines Namensgebungs- und Verbindungs-konzeptes verbessert die Einsichtigkeit und Nachvollziehbarkeit von Konzepten.

- Durch Implementationsabstraktion oder Modularisierung von Struktur, Semantik und Operationalität auf der Grundlage von Verkapselung und Scoping kann die Konzeptunabhängigkeit verbessert werden. Wichtige Methoden sind:
– das **Verstecken von Konzepten** (Sichtenbildung) (private, Gruppen- und Weltkonzepte) und
– **Abbildungsmechanismen für Sichten**.

Wir unterscheiden im Informationssystementwurfsprozeß **Konstruktionsarten**. Allgemeine Hilfsmittel zur Darstellung der einzelnen abstrakten Konstrukte sind in Anlehnung an Konstruktorkonzepte die folgenden Elemente:

- **Elementare Einheiten** zur Darstellung von Basiskonzepten,
- **Konstruktionsregeln** zur induktiven Konstruktion von komplexeren Konzepten aus bereits konstruierten oder Basiskonzepten (die meist als Konstruktionsmethodiken verstanden werden) und
- **Konsistenzregeln** wie Integritätsbedingungen und die „Normalisierung“ erlauben eine Sicherung der Qualitätsanforderungen. Einbettungsregeln ermöglichen eine Integration in den bereits vorhandenen Entwurf unter Berücksichtigung von Prioritäten, Anwendbarkeitsregeln etc.

Zur Darstellung von Strukturierung und Funktionalität können verschiedene **Repräsentationsmechanismen** gewählt werden.

Darauf aufbauend sind verschiedene **Entwurfszenario** möglich:

**Datenstrukturgetriebener Entwurf**: Es wird zuerst die Struktur der Anwendung dargestellt, darauf aufbauend die Funktionalität und die Interaktion. Dieser Zugang wird am häufigsten im Informationssystementwurf angewandt.

**Prozeßorientierter Entwurf**: Es werden zuerst die Prozesse und die erwünschte Funktionalität der Anwendung dargestellt und auf dieser Grundlage die Struktur und Interaktion. Dieser Zugang wird im Rahmen der Softwaretechnologie angewandt, er ist aber für den Datenbankentwurf in dieser Ausprägung wenig sinnvoll.

**Architekturmaßgebener Entwurf**: Es wird zuerst ein „Bauplan“ des Informationssystemes anhand der Anwendung abgeleitet. Die Architektur basiert auf **Komponenten** und **Assoziationen** zwischen den Komponenten. Es werden die einzelnen Komponenten unter Berücksichtigung ihrer Assoziationen und daraus entstehender Obligationen entwickelt.

**Interaktionszentrierter Entwurf**: Es wird zuerst der Interaktionsraum oder der Storyraum modelliert und daraus werden dann Anforderungen an die Strukturierung und Funktionalität abgeleitet. Diese Anforderungen führen zur Ableitung des Anwendungssystemes.

**Integritätszentrierter Entwurf**: Es werden zuerst die semantischen Bedingungen erfaßt. Die Struktur und die Funktionalität wird erst dann entwickelt. Obwohl dieser Zugang für den Datenbankentwurf am besten geeignet ist, existieren nur wenige Ansätze aufgrund der Komplexität im ersten Schritt.

Weitere Strategien sind möglich, wie z.B. parallele Entwicklung verschiedener Konzepte bzw. Teilkonzepte.

Orthogonal dazu sind verschiedene **Unabhängigkeitskonzepte** möglich:

- Unabhängigkeit des Endnutzers von spezifischer konzeptioneller Repräsentation und
- Unabhängigkeit der Repräsentation der Implementierung.

Diese Unabhängigkeitskonzepte sind an der Vorgehensweise zur Implementation und der **3-Ebenen-Architektur** (Endnutzerebene, Konzeptionelle Ebene, Implementationsebene) orientiert.
2 Das Abstraktionsschichtenmodell zur Modellierung

Durch Zachman wurden Ende der achtziger Jahre allgemeine Modellierungsregeln eingeführt, die mit dem Abstraktionsschichtenmodell verallgemeinert werden. Es werden in diesem Entwicklungsmodell verschiedene Dimensionen der Entwicklung unterschieden:

- Die Dimension der statischen Aspekte stellt Strukturierung der Daten und die Sichten dar.
- Die Dimension der dynamischen Aspekte soll die Funktionalität und die Interaktivität der Anwendung repräsentieren.
- In der Verteilungsdimension wird die Lokalität der Strukturen und Prozesse dargestellt.
- Die Benutzerdimension dient der Darstellung des Systemes aus Benutzersicht einschließlich der Organisationsmodelle.

Wir können diese Entwicklungsmodell um weitere Dimensionen erweitern. In der Zeitdimension wird die Entwicklung der Anwendung dargestellt. Mit der Motivationsdimension erfolgt eine explizite Darstellung der Umstände, Ziele und Motive für die einzelnen Aspekte der Anwendung.

Jede der Dimensionen verfügt über ein einfaches und eindeutiges Basismodell. Jede der Dimensionen repräsentiert genau eine Sichtweise auf die Anwendung. Ziel ist dabei im Gegensatz zu Sprachen wie die der UML daß jedes abstrakte Objekt nur einmal repräsentiert wird. Entwurfsprodukte sind aufgrund ihrer Architektur und Entwurfsgeschichte rekursiv oder iterativ aufgebaut.

Wir betrachten explizit unterschiedliche Abstraktionsschichten [Tha00, Tha03, Tha07] und integrieren die Darstellung der Architektur der Anwendung und die Versionierung explizit in die einzelnen Entwurfschritte. Damit unterscheiden wir folgende Schichten:

- die Anwendungsbereichsschicht zur Spezifikation der Ziele, der Aufgaben und der Motivation der Informationssystemanwendung,
- die Geschäftsprozeßschicht zur Spezifikation der Geschäftsprozesse, der Ereignisse, zur Grobdarstellung der unterlegten Datenstrukturen und zur Darstellung der Anwendungsstory,
- die Aktionsschicht zur Spezifikation der Handlungen, der Detailstruktur der Daten im Sinne eines Vorentwurfs, zur Darstellung eines Sichtenskeletts und zur Darstellung von Szenarien zu den einzelnen Anwendungsstories,
- die konzeptionelle Schicht zur Darstellung der Prozesse, des konzeptionellen Schemas, der konzeptionellen Sichten und der Dialoge in zusammenhängender Form,
- die Implementationsschicht zur Spezifikation der Programme, der physischen und logischen Schemata, der externen Sichten und zur Darstellung der Inszenierung.

Die Anwendungsbereichsschicht kann als strategische Schicht aufgefaßt werden. Es werden alle strategischen Entscheidungen zum Informationssystem getroffen. Sie kann selbst wieder gegliedert sein in

- die anwendungsunabhängige generelle Beschreibung (AIS) (application independent specification) des Anwendungsgebietes auf der Grundlage von Lebensfällen [ST08b], die eine Reihe von Lösungen für die speziellen Anwendungen offenläßt und die die Entscheidungen der konkreten Anwendungen nicht vorweg nimmt und
Die anwendungsabhängige Beschreibung (ADS) (application dependent specification) der spezifischen Lösungen auf der Grundlage von Anwendungsfällen (business use cases [ML05]), die in der konkreten Anwendung bereits eine spezifische Lösung vorsehen.

Diese Unterscheidung ist analog zu der klassischen MDA-Unterscheidung in PIM (platform independent model) und PDM (platform dependent model). Sie lehnt sich an die Arbeiten von [Jac06] und [Bjo06], bei der zwischen Lösung und Problem unterschieden wird und das Software Engineering in die drei Phasen

application domain description, requirements prescription and software specification unterschieden wird.

Die Geschäftsprozeßschicht wird oft auch als Anforderungsspezifikationsschicht bezeichnet. Im Rahmen dieser Schicht werden neben den Anforderungen jedoch auch konkrete Entscheidungen zur Realisierung getroffen, so daß wir diese Schicht zur Spezifikation der Anforderungen, der pragmatischen Annahmen, der Systemumgebung und der Systemorganisation und -architektur erweitern müssen. Die Aktionsschicht ist mit dem Abstraktionsschichtenmodell eingeführt worden, um eine explizite Darstellung der Anwendungsszenario vornehmen zu können. Im klassischem Systementwurf wird diese Schicht meist übergangen und zu einem späteren Zeitpunkt durch entsprechende Sichten-Suiten hinzu gefügt, wodurch ein Systembruch entsteht, den wir mit der expliziten Darstellung vermeiden können.

Abbildung 1: Das Abstraktionsschichtenmodell des Informationssystem-Entwicklungsprozesses

Das Abstraktionschichtenmodell in Bild 1 erlaubt eine Entwicklung von Informationssystemen im Zusammenhang. Wir können ein schichtenorientiertes Vorgehensmodell ebenso wie ein Modell anwenden, das sich zuerst auf einen der Aspekte orientiert.


Das Abstraktionschichtenmodell erlaubt die Darstellung der Entwicklungsresultate auf unterschiedlichem Abstraktionsniveau. Wir folgen hier im wesentlichen dem induktiven Ansatz zur Beschreibung. Damit ist jedes Resultat aus jeder Sichtweise (Strukturierung, Funktionalität, Interaktivität, Unterstützung der Interaktivität) als generelle Einheit oder Basiseinheit spezifizierbar. Resultate der Entwicklung der Informationssystemanwendung sind:

Produkte zur Darstellung der Strukturierung sollen die Strukturierung der Daten auf unterschiedlichem Abstraktionsniveau beschreiben. Wir nutzen dazu eine Separation der Spezifikation in Schemata zur Beschreibung der gesamten Strukturierung und Daten-Typen zur Beschreibung der einzelnen Struktur und der Integritätsbedingungen.

Produkte zur Darstellung der Funktionalität sollen eine Darstellung der Funktionsaspekte ermöglichen. Wir unterscheiden Workflows zur Darstellung der Folgen von Prozessen der Anwendung.

Produkte zur Darstellung der Interaktivität sollen eine Beschreibung der Anwendung aus der Sicht der Benutzer ermöglichen. Deshalb wird die Interaktivität als Raum von Handlungsabläufe der Benutzer oder ihrer Abstraktionen als Akteure, d.h. als Story-Raum beschrieben. Dieser Story-Raum fußt auf Szenen zur Beschreibung eines generellen Schrittes der Anwendung und auf Dialogschriften zur Beschreibung der einzelnen Aktionen.

Produkte zur Darstellung der Unterstützung der Verteilung sind im Rahmen von Anwendungen der Informationssysteme Sichten auf die Datenbanksysteme, Dienste zur Bereitstellung der erweiterten Sichten und deren Austauschrahmen.
Wir wollen diese Entwicklungsresultate auf unterschiedlichem Abstraktionsniveau darstellen. Wir können jeweils die Resultate mit der Abstraktionsschicht verbinden. Dann sind die Abstraktionsschichten mit folgenden Entwicklungsresultaten verbunden:

**anwendungsabhängige Beschreibung (ADS)** mit dem *Lastenheft*,

**Geschäftsprozeßschicht** mit dem *Pflichtenheft*,

**Aktionsschicht** mit der *Aktionsspezifikation* und den vier Aspekten Anwendungsschema, Nutzer-Maschine, Storyboard und Aktionssichten-Suite,

**Konzeptionelle Schicht** auf Grundlage der *konzeptionellen Spezifikation* und der Beschreibung der vier Aspekte durch ER-Schema, Workflow-Maschine, Drehbuch und Sichten-Suite,

**Implementationsschicht** auf Grundlage der *logischen Spezifikation* und einer Beschreibung der vier Aspekte durch logisches Schema, Datenbank-Maschine, Inszenierung und logische Sichten-Suite.

### 3 Ein ER-basiertes Vorgehensmodell

Wir können mit dem Abstraktionsschichtenmodell zur Entwicklung von Informationssystemen eine Reihe verschiedener Entwicklungsmodelle unterstützen:

In der **strukturierungsorientierten Entwicklung** wird zuerst die Datenbank-Struktur weitestgehend entwickelt. Darauf aufbauend werden die Prozesse und die Sichten und abschließend die Präsentationskomponente entworfen und implementiert. Diese Vorgehensweise entspricht dem klassischen Entwicklungsansatz, hat aber den Nachteil einer hohen Modifikationsrate aller vorher erstellten Dokumente.

In der **prozeßorientierte Entwicklung** wird zuerst die Funktionalität der Anwendung entworfen und prototypisch realisiert. Danach werden die entsprechenden Datenstrukturen entwickelt und abschließend die Präsentationskomponente und die entsprechenden Sichten. Dieser Zugang wird im Software-Engineering präferiert, entspricht aber selten den Gegebenheiten der Entwicklung von Informationssystemen.

**Interaktionsraum-determinierte Entwicklung**: Es werden zuerst die Stories und Szenarien der Anwendung abgenommen. Auf dieser Grundlage werden die entsprechenden Medientypen konzipiert. Damit sind die Anforderungen für die Strukturierung und die Funktionalität bekannt, so daß eine Entwicklung dieser Aspekte integriert erfolgen kann. Diese Vorgehensweise entspricht der Entwicklungmethodik von informationsintensiven Websites. Sie bedingt jedoch eine weitestgehende Erfassung aller Szenarien der Anwendung.

**Sichtenorientierte Entwicklung**: Es wird ein Skelett oder eine Architektur der Anwendung entwickelt. Die einzelnen Sichten werden schrittweise und an ihren Schnittpunkten integriert entwickelt. Darauf aufbauend können die Strukturierung, der Story-Raum und die Funktionalität entwickelt werden. Diese Vorgehensweise eignet sich besonders für gut strukturierte Anwendungsgebiete mit separierbaren Datenbeständen. Sie bedingt jedoch eine höhere Disziplin und Koordinierung bei der integrierten Entwicklung.
**Schichtenbasierte Entwicklung:** Es werden zuerst alle Aspekte auf der Anwendungsgebietskenschicht, danach auf der Geschäftsprozeßschicht, dann auf der Aktionsschicht und abschließend die Aspekte auf der konzeptionellen Schicht entwickelt. Nach Abschluß des konzeptionellen Entwurfes wird eine Transformation hin zur logischen Spezifikation vorgenommen. Dieser Zugang erfordert wenige Korrekturen im Entwicklungsprozeß und erscheint deshalb besonders geeignet. Er wird im weiteren präferiert.


Die einzelnen Schritte in Bild 2 sind die folgenden:

* **Anwendungsbereichsschicht**
  1. Entwicklung der Motivation und der Ziele der Anwendung, Informationsanalyse, Herausschälen des Gestaltungsrahmens
  2. Entwicklung des Lastenheftes zur Anwendung

* **Geschäftsprozeßschicht**
  3. Separation der Systemen in Komponenten und Entwicklung der Architektur des Systems
  4. Skizzierung des Story-Raumes, Formulierung der Interaktivität für das Pflichtenheft
  5. Skizzierung der Sichten-Suite für die einzelnen Komponenten, der Dienste und des Austauschrahmens, Formulierung der Verteilung und Strukturierung für das Pflichtenheft
  6. Spezifikation der Business-Prozesse, Formulierung der Funktionalität für das Pflichtenheft

* **Aktionsschicht**
  7. Spezifikation der Szenario der Anwendung
  8. Beschreibung der Haupttypen der einzelnen Sichten und deren Assoziationen
  9. Entwicklung der Integritätsbedingungen und deren Erzwingungsstrategie
  10. Spezifikation der Benutzeraktionen, Rollen, Skizzierung der Content-Typen
  11. Spezifikation der Qualitätsanforderungen und deren Umsetzung im System, Entwicklung von Sicherungsstrategien

* **Konzeptionelle Schicht**
  12. Spezifikation des Story-Raumes
  13. Spezifikation der Akteure, ihrer Portfolio, Rollen, Rechte, Profile
  14. Spezifikation der Sichten-Suite, der Dienste und Austauschrahmen
  15. Entwicklung der Workflows
Abbildung 2: Schritte in unserem Vorgehensmodell
16. Kontrolle der Content-Typen anhand von Content-Objekten, Validierung der statischen Semantik, Kontrolle der Integritätserzwingung
17. Spezifikation der Szenen, der Dialogschritte, der Bedingungen für die Stories, der Handlungsumgänge
18. Spezifikation der Content-Typen-Suite, der notwendigen Funktionalität zu deren Unterstützung
19. Modulare Verfeinerung der Datentypen
20. Normalisierung der entwickelten Datentypen
21. Kontrolle des Story-Raumes anhand der Szenario, Ableitung weiterer möglicher Szenario, Blockierung unerwünschter Szenario, Ableitung der Verlinkungs- und Navigationsstruktur, Kontrolle der unterstützten Funktionalität
22. Spezifikation der Funktionalität, Kontrolle des Verhaltens der Anwendung, Abstimmung der Unterstützung für Dienste, Austauschrahmen, Kollaboration
23. Integration der Sichten-Suite anhand der Architektur des Systems, Auflösung der Entwurfsobligationen

* Implementationsschicht
24. Transformation der konzeptionellen Modelle in logische Modelle zur Darstellung der Strukturierung, Funktionalität, Interaktivität und Verteilung
25. Restrukturierung und Optimierung auf der Grundlage von Performanzbetrachtungen und des Tuning
26. Ableitung des Dienstverwaltungssystemes, der Protokolle und der Funktionen zur Unterstützung der Verteilung
27. Transformation der logischen Modelle in physische Modelle des DBMS

4 Anwendung unserer Methodik


Literatur


Towards ASM Engineering and Modelling

Bernhard Thalheim and Peggy Schmidt

Christian-Albrechts-University Kiel, Department of Computer Science, 24098 Kiel, Germany
thalheim | pesc@is.informatik.uni-kiel.de

Abstract

The ASM approach has gained a maturity that permits the use of ASM as the foundation for all computation processes. All known models of computation can be expressed through specific abstract state machines. These models can be given in a representation independent way. Stepwise refinement supports separation of concerns during software development and will support component-based construction of systems, thus providing a foundation of new computational paradigms such as industrial programming, programming-in-the-large, and programming-in-the-world.

Despite the theoretical and application maturity a modelling theory for ASM specifications does not exist. Pragmatism and methodologies are necessary whenever larger systems have to be specified. We develop a number of principles and approaches to ASM specification that allow one to develop modular and surveyable ASMs. Our approach is based on the Turbo ASMs, abstraction layers, and on refinement. ASM engineering is based on a well-defined methodology that promises to be manageable.

1 Modelling of Applications Based on ASM

1.1 Properties of Modelling and Kinds of Abstraction

Modelling is one of the most difficult tasks in software engineering. It aims at a representation or simulation of reality and at identification of a particular model. The given application is subject to analysis by modelling if it can be described in terms of expressions in the language used for modelling. The model is a result of modelling. It relates things \( D \) under consideration with concepts \( C \). This relationship \( R \) is characterised by restrictions \( \rho \) to its applicability, by a modality \( \theta \) or rigidity of the relationship, and by the confidence \( \Psi \) in the relationship. The model is agreed upon within a group \( G \) and valid in a certain world \( W \). Stachowiak [Sta92] defines three characteristic properties of models: the mapping property (have an original), truncation property (the model lacks some of the ascriptions made to the original), and pragmatic property (the model use is only justified for particular model users, tools of investigation, and period of time). In [KT06] is additionally considered the extension property. The property allows models to represent judgments which are not observed for the originals. In computing, for example, it is often important to use executable models. Finally, the distortion property is often used for improving the physical world or for inclusion of visions of better reality.

Software engineering uses a number of principles that refine different kinds of abstraction [Tha00] such as construction abstraction, context abstraction and refinement.
abstraction. Construction abstraction uses the principles of hierarchical structuring, constructor composition, and generalisation. Refinement abstraction uses the principle of modularisation. Hierarchical structuring uses the decomposition of software into subparts in such a way that constituents form a tree. Modularisation encapsulates components and uses interfaces for exclusive communication of components with the environment. Constructor composition depends on the constructors used [Tha05]. Additionally, principles such as well-founded structuring may be applied. The last principle requires that only such constructors are applicable (sequence, bounded iteration, choice, etc.) for which the compositionality principle is preserved and semantics can be derived based on the inductive construction.

1.2 Challenges of Modern Software Engineering

Software engineering is still based on programming in the small although a number of approaches has been proposed for programming in the large. Programming in the large uses strategies for programming, is based on architectures, and constructs software from components which collaborate, are embedded into each other, or are integrated for formation of new systems. Programming constructs are then patterns or high-level programming units and languages. The next generation of programming observed nowadays is programming in the world within a collaboration of programmers and systems. It uses advanced scripting languages such as Groovy with dynamic integration of components into other components, standardisation of components with guarantees of service qualities, collaboration of components with communication, coordination and cooperation features, distribution of workload, and virtual communities. The next generation of software engineering envisioned is currently called programming by composition or construction. In this case components form the kernel technology for software and hardware.

Software development is mainly based on stepwise development from scratch. Software reuse has been considered but has never reached the maturity for application engineering. Software development is also mainly development in the small. Specifications are developed step by step, extended type by type, and normalized locally type by type. Software engineering is still be considered as handicraft work which requires the skills of an artisan. Instead, we need techniques for this century [Boe06]. Classical software development methods are mainly appropriate for programming in the small and combination of such programs into a program system. The ASM approach has the expressivity to handle also programming in the large together with programming in the small. Engineering in other disciplines has already gained the maturity for industrial development and application we need to reach.

Software engineering can be based on the trilogy consisting of the application domain description, the requirements prescriptions, and finally the systems specifications [Bjo06,Hei96]. This approach extends modern software engineering approaches by explicit consideration of the application domain.

Advanced applications such as web information systems require novel specification and development methods since their specification is oriented towards systems that are easy and intuitively to use. [Tha03,ST05] extend these approaches by (1) explicit con-
1.3 Achievements of the ASM Approach

The ASM method nicely supports high-level design, analysis, validation and verification of computing systems:

- ASM-based specification improves industrial practice by proper orchestration of all phases of software development, by supporting a high-level modelling at any level of abstraction, and by providing a scientific and formal foundation for systems engineering. All other specification frameworks known so far only provide a loose coupling of notions, techniques, and notations used at various levels of abstraction. By using the ASM method, a system engineer can derive a general application-oriented understanding, can base the specification on a uniform algorithmic view, and can refine the model until the implementation level is achieved. The three ingredients to achieve this generality are the notion of the ASM itself, the ground model techniques, and the proper treatment of refinement.

- Abstract state machines entirely capture the four principles [ZT04] of computer science: structuring, evolution, collaboration, and abstraction.

This coverage of all principles has not been achieved in any other approach of any other discipline of computer science. Due to this coverage, the ASM method underpins computer science as a whole. We observe the following by comparing current techniques and technologies and ASM methods: ASM are running in parallel. Collaboration is currently mainly discussed at the logical or physical level. Evolution of systems is currently considered to be a hot but difficult topic. Architecture of systems has not yet been systematically developed.

- The ASM method is clearly based on a number of postulates restricting evolution of systems. For instance, sequential computation is based on the postulate of sequential time, the postulate of abstract state, and the postulate of bounded exploration of the state space. These postulates may be extended to postulates for parallel and concurrent computation, e.g., by extending the last postulate to the postulate of finite exploration.
1.4 Plan of the Paper

We are completely aware of the complexity of the modelling problem and do not expect that it can be solved within a single conference paper. Therefore, we restrict our efforts to modelling instruments of the ASM method, and to separation of concerns into development layers. In Section 2, choices for modelling are discussed: modularisation, agent orientation, styles and pattern. The section concludes with general properties. These choices are illustrated in Section 3 for a specific modelling method: layered ASM modelling. Due to space limitations we do not use sophisticated examples. We also do not discuss architectures of ASM machines. Section 4 concludes the paper with a discussion on future work.

2 ASM Modelling Alternatives

In [KT06] a general approach is proposed to modelling that starts with a clarification of the properties of modelling and of the kinds of abstraction that are considered and with an elaborated and reasoned selection of the modelling language that includes detailed knowledge of deficiencies of this language and therefore avoids the Sapir-Whorf hypothesis [Who80]. We extend this framework by an application-driven choice of architecture and platform, by a collection of modelling styles, and by orchestration of modelling techniques such as pattern. In this case, we shall be able to derive properties of the modelling process.

2.1 Modularisation

Modular modelling supports information abstraction and hiding by encouraging and facilitating the decomposition of systems [BM97] into components and their modular development based on a precise definition of interfaces and the collaboration of components through which the systems are put together. Implicit modularisation can be achieved by introduction of name spaces on signatures. Explicit modularisation offers a better understanding of structure and architecture of systems and thus supports consideration of evolution of systems and of collaboration of systems.

Modularisation offers a number of advantages: separation of concerns, discovery of basic concepts, validation and verification of development, efficiency of tool support, and - last but not least - scoped changes. The last advantage of modularisation is based on an explicit framing of development to a number of elements while preserving all other elements in its current form. We model this impact by introducing name spaces on signatures.

Typically, small submachines capture smaller models that are easier to understand and to refine. Small models can better be ascertained as to whether we need to apply refinements.

Modularization is a specification technique of structuring large specifications into modules. It is classically based on structural and functional decomposition [BS00]. We additionally consider control decomposition. Modules form a lattice of associated submachines having their own states and their own control.
Modularisation is based on implementation abstraction and on localization abstraction. Implementation abstraction selectively hides information about structures, semantics and the behavior of ASM concepts. Implementation abstraction is a generalization of encapsulation and scoping. It provides data independence through the implementation, allowing the private portion of a concept to be changed without affecting other concepts using that concept. Localization abstraction "factors out" repeating or shared patterns of concepts and functionality from individual concepts into a shared application environment. Naming is the basic mechanism for achieving localization. Parametrisation can be used for abstraction over partial object descriptions. We use the name space for handling localisation abstraction.

2.2 Agent-Oriented Specification

An ASM submachine consists of a vocabulary and a set of rules. In this case, any clustering of rules and of elements from the vocabulary may define a submachine. Turbo ASM [BS03] capture our notion of a submachine by encapsulating elements of the vocabulary and rules into an ASM. They hide the internals of subcomputations within a separate ASM. The submachine has its own local state and its own interface.

The set of functions of each submachine can be separated into basic and derived functions. Basic functions may be static functions or dynamic functions. Classically [BS03] dynamic functions can be classified as in(put) functions, out(put) functions, controlled or local functions that are hidden from the environment, and shared functions that are visible to the environment. A similar classification can also be applied to basic static functions. They are either functions only used by a its own machine or read by several environments. We thus extend the notion of shared and controlled functions to static functions as well. We do not use derived static functions since they can be considered as syntactic sugar. We differentiate these functions according to their role in Figure 2 which displays the functions internal for an agent ASM. A similar classification can be developed for functions external to an agent. An agent ASM consists of all functions that assigned to the agent and of rules that are assigned to the agent and that use only those functions assigned to the agent.

![Fig. 2. The Kinds of Internal Functions for Agent ASMs](image-url)
Static functions may also be local functions. They are not updated by any submachine. [BM97] distinguish derived function to whether these functions are monitored functions, controlled functions, or shared functions. Typically, derived functions are functions that do not exist on their own right, but may be dynamically computed from one or more base functions. They provide a powerful and flexible information hiding mechanism. Updates made in the base functions that affect the derived function are immediately reflected in derived functions.

We may additionally assume that derived functions are allowed to update dynamic functions. In this case, dynamic functions may be used as a security mechanism, as an access mechanism, and as a simplification mechanism that allows to use complex derived functions in rules instead of complex computations in rules.

### 2.3 Perspectives and Styles of ASM Modelling

Different modelling perspectives can be distinguished:

1. The *structure-oriented perspective* focuses on structural description of the ASM. Sometimes, the structure-oriented perspective is unified with the semantic perspective. In this case, design of the structure is combined with design of invariants.
2. The *behavior-oriented perspective* is concerned with the behavior of the ASM during its lifetime. It can be based on event approaches or on Petri-net approaches and predicate transition systems.
3. The *process-oriented perspective* is concerned with the operation of the system.

The structure-oriented perspective is often used for data-intensive applications. Almost all recognized database design approaches are based on the structure-oriented perspective. The process-oriented perspective uses approaches considered in software engineering. The behavior-oriented perspective is a high-level descriptive approach to an integrated specification of the vocabulary and rules.

Modelling styles provide a very abstract description of a particular set of general characteristics of a model. Different constructional notations may be useful for describing a machine. We use the Turbo ASM approach for component or submachine description. Typically, the role of the components of the system follow the rules specified by the style. The modelling style explains the structure, the abstraction and grouping of the elements. Parts of the ASM may follow different modelling styles.

The style of modelling is a specification of the high level structure and organisation of ASM modelling. The structure describes the handling of elements of the vocabulary, the topology or relationships between elements, the semantical limitations for their usage, and the interaction mechanism between the elements such as blackboard, submodule calls, etc. The organisational style describes relevant local and global structures, the decomposition strategy, and control mechanisms between parts of the ASM machine. The organisational style is based on the architectural style. It is our aim to maintain and to preserve the strategy over the life cycle of the system.

The perspective and the style result in *strategies* that are use for step-wise development of specifications. The different strategies [Tha00] based on the structure-oriented perspective are sketched in Figure 3.
2.4 Pattern of ASM Vocabulary and Rule Descriptions

The notion of pattern originates from traditional architecture and denotes a general repeatable solution to a commonly occurring problem in software design. Typically, patterns are instantiatable expressions and can be transformed directly into specifications.

The vocabulary description may follow a number of different patterns. Structure-oriented strategies may apply a number of different pattern, for instance the following:

Compacting patterns integrate functions and represent them through one function. They provide a compact representation. For instance, the file specification in [Stä04] uses

\[ \text{argFile} : \text{isActive} \rightarrow \text{isFile} \]
which assigns files to agents.

The compacted function

\[ \text{argName} : \text{isActive} \rightarrow \text{string} \]

assigns a new file name to the file to be created or assigns a new name to the file to be moved. In both cases, an agent has only one task and is thus assigned to one file by \text{argFile} for each file operation. The function compacts the functions

\[ \text{argName}_{\text{Create}} : (\text{isActive} \rightarrow \text{isFile}) \rightarrow \text{string} \]
\[ \text{argName}_{\text{Move}} : \text{isFile} \rightarrow \text{string} \]

Typing patterns divide the vocabulary into types and define functions within a type or within associations among types. The vocabulary can be divided into types. For instance, the file specification is separated from the activities of agents.

Unfolding patterns provide all functions that are associated with a domain. In our example, all functions that are definable for agents are specified.

Union patterns use the most general range for functions. For instance, an agent has a main parent directory that is essentially a file. Therefore, [Stä04] specifies

\[ \text{argParent} : \text{isActive} \rightarrow \text{isFile} \]

and tests in rules whether the result of \text{argParent} is a directory.

Each of these patterns has its advantages. Compacting and unfolding pattern allow convenient rule description. Refinement is supported by typing patterns. Unfolding pattern
lead to high redundancy that must be effectively supported. Union patterns avoid the covariance / contra variance problem but lead to problematic rules.

These pattern result in description styles. Typical description styles for structure-oriented perspective are

– predicative representation that uses a Boolean functions for the vocabulary or predic-icates and
– functional representation that uses functions with ranges of arbitrary domain types is appropriate for specification of event systems.

The structure-oriented perspective is typically based on a predicative pattern of specification.

The description of the state space can be either given based on open world pattern or a closed world pattern. The closed world pattern allows only those values that are explicitly given. The values are typically given through ground-term algebras, e.g., enumeration types. The open world pattern allows to extend the state space whenever this is necessary.

The description of terms, variable assignment, formulas, and interpretation is typically based on the canonical specification pattern of mathematical logics.

The rule description also may follow patterns. Typical ASM patterns are the following:

**Event-condition-action patterns** are based on a separation of the state space into event states and other states. **Control state patterns** are based on explicit usage of control states. These states are used to separate activity of rules into those that are applicable and those that are not applicable. **Error patterns** can be combined with any rule. They may be folded into the activities of the rule or into the condition of the rule. **State transition patterns** are used for transition to a new state from the current state. They are typically folded into activities of rules. **Macro patterns** are parameterised rules that allow to reuse fragments of ASM [SSB01].

We can use almost all software engineering pattern within ASM specifications. Therefore, the pattern list might be rather large. Figure 4 survey different kinds of pattern for rules.

![Fig. 4. Kind of ASM Rule Pattern](image)

Pattern instantiation requires fitting of values to the parameters. This fitting is specified through context conditions. For instance, values used in conditions must fit into domains that are valid for the states potentially given for the vocabulary.
2.5 Pattern for Invariants

Invariants, e.g. integrity constraints in database applications, are used to define semantics of applications. We know different pattern for their specification:

- **Operational representation** of invariants incorporates invariants into the programs or rules. The invariant enforcement mechanism may be hidden because of control conditions or to the specification of actions.

- **Descriptive representation** uses explicit specification and refinement obligations. These descriptions are combined with the specification of invariant enforcement:
  - Eager enforcement maintains invariants based on a scheduling mechanism for maintenance of invariants. Transactional systems are typical scheduling mechanisms. They bind invariant enforcement to programs.
  - Lazy enforcement maintains invariants in a delayed mode. Inconsistency is temporarily tolerated. This tolerance reduces some of the cost of enforcing invariants within large structures.
  - Refusal enforcement maintains invariants by rollback of all activities since the last consistent state and by executing a subset of activities. Partially ordered runs are based on refusal enforcement.

Depending on the pattern chosen invariant handling is varies. If we choose an implicit invariant handling then any change applied to the current ASM must explicitly consider all invariants and must be entirely aware of the effects of these. Therefore this pattern is the most inefficient for early design phases. This pattern is however applicable during implementation if later revision is going to be based on a more general ASM.

The completeness of invariant specification is a dream that is never satisfied. Sets of invariants are inherently open since we cannot know all invariants valid in the current application, we cannot envision all possible changes in invariant sets, and we cannot choose the most appropriate selection of invariants from which all other invariants follow. Therefore, we use a separation into

- hard (or iron) invariants that must be preserved and which are valid over a long time in the application and
- soft invariants that can be preserved or are causing later corrections or which are not valid for a longer time in the application.

2.6 ASM Modelling Assumptions

The unique name assumption requires that elements with different names are different. If we need to use the same name for different purposes then we use name spaces if a unique identification is needed. We distinguish between identification through call by name, call by reference, call by value, call by value-result, call by constant value, and call by value name. Identification cased on call by reference and call by name are the most universal. Call by constant value is supported by system generated identifiers. Call by value supports value identifiability. The closed world assumption presumes that the only possible elements are those which are specified. The domain closure assumption limits the elements in the language to those that can be named through the vocabulary, the states of the vocabulary or the rules.
Two additional assumptions we may apply are the *unique meaning assumption* and the *universal machine assumption*. The first assumption postulates that any function or rule of the ASM has the same meaning despite modularisation. The second assumption postulates that the behaviour of the entire ASM can be defined by composition applied to the submachines.

Due to the variety of choices we might use additional assumptions for the development. The most general *architectural assumption* is the possibility of *layering* a system into sub-systems. We might use other assumptions such as common data pools, transactional systems providing an exclusive write to a location for one sub-system and a guided read with un-read to this location for all other subsystems. The use of shared functions determines whether a system consists of strictly separated components that do not have shared functions or consists of a system of components with overlapping, i.e., shared functions.

### 2.7 Properties of ASM Modelling

The software development or generally the modelling process is intentionally or explicitly ruled by a number of development strategies, development steps, and development policies. Modelling steps lead to now specifications to which quality criteria can be applied. Typical quality criteria are completeness and correctness in both the syntactical and semantical dimensions. We assume that at least these four quality criteria are taken into consideration. The modelling process can be characterised by a number of (ideal) properties:

**Monotonicity:** The modelling process is monotonic, if any change to be applied to one specification leads to a refinement. It thus reflects requirements in a better form.

**Incrementality:** A modelling process is iterative or incremental if any step applied to a specification is only based on new requirements or obligations and on the current specification.

**Finiteness:** The modelling process is finite if any quality criteria can be checked in finite time applying a finite number of checks.

**Application domain consistency:** Any specification developed corresponds to the requirements and the obligations of the application domain. The appropriateness can be validated in the application domain.

**Conservativeness:** A modelling process is conservative if any model revision that cannot be reflected already in the current specification is entirely based on changes in the requirements.

Typical matured modelling processes are at least conservative and application domain consistent. Any finite modelling process can be transformed into a process that is application domain consistent. The inversion is not valid but depends on quality criteria we apply additionally. If the modelling process is application domain consistent then it can be transformed in an incremental one if we can extract such area of change in which consistency must be enforced.
3 Layered ASM Modelling

We elaborate one kind of ASM modelling in more detail. Layered ASM modelling is based on modularisation and on architectures of the ASM. The language layering approach we use has already been reported in a similar form in [Wal97].

3.1 Assumptions of Layered ASM Modelling

Layered ASM modelling is based on the architectural assumption that the system can be separated into components and a general layering is achievable. We base layered ASM modelling on the unique name assumption, the domain closure assumption, and the universal machine assumption. We may use the closed world assumption and the unique meaning assumption. Layered modelling is not restricted to the last two assumptions.

In general, we use architecture-driven development that starts first with the prescription of the architecture pattern and style. The agent-oriented specification of ASM allows the development of a system as a collaborating society of sub-systems. This society uses shared functions where the sharing is based on contracts for the usage of these functions, on workflows that describe the cooperation among these sub-systems, and on implicit communication based on the locations for these functions [ST07]. We may use different views of the same architecture [Sie04] such as technical views displaying the modules with their functionality, application views displaying activity zones depending on the stage of the application, infrastructure views displaying the dependence of the system from its infrastructure and supporting systems, or the context view that considers the whole organisational, story and application context.

3.2 Vocabulary Modelling

The five properties of modelling are: mapping, truncation, pragmatic, extension and distortion properties. These properties govern abstraction. We consider six different aspects for vocabulary modelling: intention, usage, content, functionality, context, and realisation. The intention aspect is a very general one centered around a mission statement for the system. The primary question is: what is the purpose of the system? Once some clarity with respect to the intentions of the system has been obtained, it is important to anticipate the behaviour of the users. The content aspect concerns the question: Which information should be provided? The functionality aspect is coupled with the question, whether the system should be passive or active. The context aspect deals with the context of the system with respect to society, to time, to expected users, to history of utilisation and to the paths of these users through the system. The realisation aspect concerns the final implementation.

Vocabulary modelling must cover all these aspects in a proper form. It may be based on partial order-sorted signatures. Depending on the choices we made for modularisation, separation of concerns through agent-oriented specification, perspectives and styles, pattern, and properties of modelling itself we can use different representations and conventions. Typical conventions are naming and binding conventions.

Layered ASM modelling can be based on typing pattern. We are additionally interested in incrementality and application domain consistency. We additionally assume
value-identifiability for each element of the state. This assumption allows to define equality and inequality for each pair of elements of the state. Therefore we develop a multi-layered vocabulary modelling approach that uses the following ingredients:

**Domain types** are used for introducing the set of states envisioned. The superuniverse is the union of all states. Beside basic value types such as string, int, float etc. we assume domain types

```
BOOL, NULL, Ø, ID
```

consisting of the truth values true, false, of the value undef, of the empty set, and of a set of identifiers, correspondingly.

Domain types can be complex types that are inductively constructed from basic types by applying constructors such as (Cartesian) product, set, list, and multi-set constructors.

Domain types can be labelled by an abstract name. Domain types are typically order-sorted. We restrict the partial order to lattices or Brouwerian algebras.

**Abstract types** are denoted by a triple consisting of an abstract name denoting the abstract type, of a domain type used for values, and a set of invariants limiting the state space that can be used for interpreting the abstract type. The set of invariants may be empty. In this case we omit the third part of the triple. If the abstract name coincides with the domain type name then it can be omitted as well.

**Predicates** are specified by their name, an arity, a sequence of abstract types, and a set of invariants, where the length of the sequence coincides with the arity. Invariants limit the state space that can be used for interpreting the predicate. It can be empty. Predicates are interpreted on the basis of set semantics.

**Functions** are specified by their name, an arity, a sequence of abstract types used for the domain of the function, an abstract type used for the range of the function, and a set of invariants where the length of the sequence coincides with the arity. Invariants limit the state space that can be used for interpreting the function. It can be empty. Functions are interpreted on the basis of set semantics for their domain.

Predicates and functions are often partial. Due to application domain consistency we assume that each abstract type has a natural meaning in the application domain. We also may assume that labels have a natural meaning in the application domain or are used as an abstraction for convenience of specification. We use the unique name assumption for all labels, types, predicates and functions. It is convenient to assume that abstract names and the superuniverse are disjoint.

Additionally we support the introduction of derived notions:

**View functions** are derived functions. They can be virtual or materialized. Virtual view functions are computed whenever this is necessary. They are not stored in the ASM. View functions can be given in

- an explicit form by their introduction in the vocabulary and
- an implicit form by their introduction in rules using the let, choose, for all and where introductions. We may distinguish

  - **transient view functions** that are used for definition of values in let and choose rules and may be directly removed whenever a value has been chosen and
• collector view functions that are used for parallel execution in forall rules and in where introduction and must have a lifespan that last over the entire rule computation.

Virtual view functions are the view functions typically used. In this case, it it assumed that the view is computed first before applying a rule, e.g. in a where introduction.

Clusters functions are disjoint unions of functions. They are well known in programming languages and are mainly used as syntactic sugar since generalisation of functions by combination eases the treatment.

Control state functions are often introduced in rules through conditions that use a certain control state as the enabler or disabler of the rule.

Special purpose functions are used for default functions and exception functions.

Each of the elements of the vocabulary has its scope that consists of the space of all locations that are used for the functions and of all rules that use these elements.

All these derived notions may be used inductively for the construction of other derived functions. Therefore, the vocabulary becomes layered depending on the construction.

Additionally, we need to consider metadata describing the specific purpose of elements of the vocabulary. They represent the content and the meaning of the functions. The meaning can be partially described by the name if the wording used has a mini-semantics. Metadata also include technical data that guide refinements applied to the vocabulary. Metadata may be partially given through a glossary or thesaurus.

Finally, a well-developed specification uses naming conventions for readability.

We distinguish between

sketchy specification of the vocabulary that uses an implicit definition of domain types and of abstract types and that specifies predicates and functions through declaration of the mappings and

sophisticated specifications of the vocabulary with detailed description of any element of the vocabulary.

The typical specification of the vocabulary uses a specification approach between these extremes. Whenever refinements need to be made then sophistication is necessary for the verification of refinement correctness.

3.3 State Space Modelling

State and vocabulary are two sides of the same coin and must be developed in a co-design process. We distinguish two extremes:

– Orientation to most general domain types: We use the most general domains for the specification. If we need specific domains then we can use unary predicates. The exclusive utilisation of most general domain types leads to complex invariants for values with specific meaning and for relations among the values and to extensive specification of various ‘exceptions’. The refinement is simpler but the burden for correct functioning is transferred to rule specification.
- Tight coupling of domain types and abstract types: Any abstract type is associated with the most specialised domain type. Specialised domain types are often associated to a specific meaning in the application. Refinement becomes more difficult. At the same time, rule specification can concentrate on the essentials. Exception due to weak domain types do not appear.

Enumeration states are often used for control states. These control states separate transitions and lead to implicit clustering of rules. Control states have a implicit scope that is defined by their utilisation in conditions and by transfer assignments in rules from or to another control state. Rules that are enabled by a control state form a submachine or a module. The transfer from one control state to another control state is represented by a state transition graph.

### 3.4 Agent-Based Modularisation

We combine modularisation and agent-oriented specification. Each function and each predicate may be visible or may not visible to an agent. Any agent has its vocabulary share. This vocabulary share states whether the given predicate or function is an in element (a monitored function or predicate), a controlled element, a shared element or an out element. A function or predicate may also be external (denoted by $E$) for an agent.

Therefore, given a set $A$ of agents and a vocabulary $V$, the vocabulary share is given by a function

$$\text{share} : A \times V \rightarrow \{E, I, O, C, S\}$$

assigning agents their kind of vocabulary use.

If an agent uses a dynamic function or predicate as its controlled function then no other agent can use it. If an agent uses a dynamic function or predicate as its in or out element then we should require that another agent uses this function or predicate as an out or an in element, correspondingly. We may additionally require that dynamic functions and predicates of the vocabulary are partitioned into in, out, controlled, and shared functions. Typically, we require that the shared dynamic function predicate are consistently assigned, i.e., the function or predicate is either external or shared for any agent that uses it internally. The same restriction can be made for static functions. This restriction is a variant of the well-known open-closed principle [Mey88] for vocabulary in layered ASM modelling.

According to Figure 2 we restrict static functions and predicates to controlled and shared share. We also exclude using derived functions and predicates as indirectly out functions.

Derived notions may be restricted to be used only as means for the simplification of vocabulary definition or as services provided by agents. In the later case they will have the same behaviour as out functions and predicates. They may be used by other agents.

It is sometimes convenient to use the elements of the vocabulary in a mixed form. For instance, a dynamic function is an in function for one agent and a shared function for other agents. We avoid this approach for layered ASM modelling since it can be a source for confusion.
Layered ASM modelling also aims to establish a clear definition of the kind of sharing. We introduce an abstract type \textit{right} that denotes the rights an agent may have for updating the shared elements and an abstract type \textit{obligation} that denotes the obligations an agent must follow if this element is a shared element for the agent. We introduce the contract for an agent by assigning rights and obligations to agent for each shared function or predicate by a partial function

\[
\text{contract} : \mathcal{A} \times \mathcal{V} \rightarrow \text{right} \times \text{obligation}
\]

that assigns the right and the obligation to each agent for each function or predicate the agent shares. It is often convenient to use the contract function as a dynamic function that can be changed by an agent that acts in the role of a controller or scheduler. In this case we can assign to an agent an exclusive update right while excluding other agents from writing. These agents have in this case only read rights. This approach eases introduction of transactional systems.

Additionally, we may use a general agent \textit{Main} for the main ASM.

### 3.5 Modular Rule Modelling

Rules are mainly specified based on different condition-action patterns, e.g., event-condition-action pattern, control state pattern, state transition pattern. The basic specification of a rule is extended by

- \textit{state dependence} that describes whether a rule can be invoked in dependence of certain conditions on a state,
- \textit{access environment} that determines where the invocation of a rule may appear, and
- \textit{control guards} that restrict invocation of a rule and are handled by a controller ASM.

The extension has been introduced for convenience. It can be expressed by an ASM that is generically added to the given ASM. We also use this extension for explicit treatment of conflicts in update sets of rules. This extension does not limit the application of partial ordered runs but allow an explicit treatment of such updateset conflicts that must be treated.

State dependence condenses conditions on the existence of certain objects in the state, value conditions for the invocation of a rule, and collaboration conditions that relate the given rule to the invocation of other rules. Collaboration conditions can also be used for explicit synchronisation of parallel rule invocation. State dependence specification supports subject-oriented programming that focuses on capturing different subjective perspectives on a single object model. It basically allows composing applications out of “subjects” (partial object models) by means of declarative composition rules.

The access environment specification contains the views on a state that are initialised when a rule is going to be executed and the internal functions that are used for execution of the rule. Each of the rules has its \textit{scope} that consists of the space of all functions and predicates that are used in the rule. The scope contains all parameters of the rule. The scope can be used to bind an element of the vocabulary to all rules have this element within its scope. This inversion is called \textit{vocabulary element scope}.

We assume the validity of the domain closure assumption for the vocabulary used in
rules. Any name used in a rule must be either a value or a type, function, predicate or view given by the vocabulary. We may allow implicit views. We also assume the open-closed principle for rules in layered ASM modelling: *If an agent is assigned to a rule then the scope must only contain such elements of the vocabulary on which the agent has an internal share.*

Control guards allow one to avoid inconsistent update sets. We use a partition of control states for separation of runs of abstract state machines. Control guards may be used as entry guards that restrict invocation of a rule and accept guards that restrict updates of a rule. The allow one to express rely-conditions and guarantee-conditions by both pre- and post-conditions. Rely conditions state what can be tolerated by the party. Guarantee-conditions record the interference that other processes will have to denote with if they are allowed to run in parallel. We envision in this paper that these conditions can be generalized to a specific style of *assumption-commitment specification*. [Wal97] and [SSB01] use exception types which are a very specific type of control guards. Control guards may also be used for introduction of break and continue statements for a rule with temporary lock and wait views.

Rules may be combined to form *macros*, which may be reused by other submachines. Macros support intentional programming and aspect-oriented separation of functionality. They can be generalised for adaptive programming, generative programming, and pattern-based development. Intentional programming provides an extendible programming environment based on transformation technology and direct manipulation of active program representations. New programming notations and transformations can be distributed and used as plug-ins in a play-in/play-out engine [HM03]. Aspect-oriented programming improves the modularity of designs and implementations by allowing a better encapsulation of cross-cutting concerns such as distributed transfer, synchronization, data traversal, tracing, caching, etc. in a new kind of modularity called “aspects”. Generative programming aims to increase the productivity, quality, and time-to-market in software development thanks to the deployment of both standard component and production automation. System families are developed rather than single systems. Generative programming uses government and binding [BST06].

4 Concluding and Plans for Evolution of ASM Modelling

ASM modelling can be based on a rigorous science of development. Development uses (a) separation of concerns as a mechanism to reduce complexity and manage change and uses (b) abstraction as a way to express models, separate levels of detail, and establish relations among the levels. This paper is a contribution to systematic development of large systems. We analysed in detail layered development for software systems. Layered ASM modelling is based on a number of postulates: inductivity, compositionality, pragmatic assumptions (unique name, unique flavour, full-fledged domains, non-triviality of structures of associations, strict hierarchical structures, non-triviality of identification), closed specifications, explicit semantics and identifiability of objects. Development is based (1) on deselection of non-interesting parts of the application and of mis-concepts that are not intended and (2) on alignment and negotiation in the syntactical, semantical, pragmatical and social dimensions. Each of these dimensions has its rules.
4.1 Treatment of Over-Specification

[HT00] discussed the downside of over-specification. A good specification contains a protocol for future extension and a portfolio for the current implementation. It is a contract with the application stakeholder and an unambiguous description of the application domain. Therefore, it seems that the specification must be as complete as only possible. This ‘completeness’ leads to the over-specification. Formal methods should not be misused for a hyper-detailed description of an application but should provide robustness against other interpretations and understandings, against changes in the application itself or within the computing environment, against evolution of the application domain, and against multiple styles of the specification. A hyper-detailed specification suffers from the straitjacket effect that limits the flexibility of specification.

The ASM methods offers executability of the specification and thus treats the strait-jacket effects and supports robustness. The over-specification problem can however been solved by explicit introduction of checkpoints that allow to overcome the dangers of over-specification. A check point measures the specification itself. These measures might be build in a similar form as metrics. One possible measure could be the vocabulary complexity of rules. We might use a threshold value which should not be exceeded for any rules. This threshold value can be based on the scope of the rules.

The modeling framework can be mapped to the Java Modelling Language [JP01]. It uses an explicit specification of contracts by pre- and postconditions, exception handling and compensation. JML is supported by a number of tools such as ESC/Java2, JACK, Daikon, KeY, and AutoJML. JML is supporting programming but not yet supporting modelling and development. Therefore, our framework may be used to enhance JML. The modelling results are going to be mapped to JML constructions. We should however not overload specification by programming constructs.

4.2 Deriving Plans and Primitives for Refinement

The perspectives and styles of modelling rule the kind of refinement styles. As an example we consider structure-oriented strategies of development depicted in Figure 3:

- **Inside-out refinement:** Inside-out refinement uses the given ASM machine for extending it by additional part. These parts are hocked onto the current specification without changing it.
- **Top-down refinement:** Top-down refinement uses decomposition of functions in the vocabulary and refinement of rules. Additionally, the ASM may be extended by functions and rules that have not yet been considered.
- **Bottom-up refinement:** Bottom-up refinement uses composition and generalisation of functions and of rules to more general or complex ones. Bottom-up refinement also uses generation of new functions and rules that have not yet been considered.
- **Modular refinement:** Modular refinement is based on parqueting of applications and separation of concern. Refinement is only applied to one module and does not affect others. Modules may also be decomposed.
- **Mixed skeleton-driven refinement:** Mixed refinement is a combination of refinement techniques. It uses a skeleton of the application or a draft of the architecture. This
draft is used for deriving plans for refinement. Each component or module is developed on its own based on top-down or bottom-up refinement.

These different kinds of refinement styles allow one to derive plans for refinement and primitives for refinement.

The refinement framework of [Bör03, Sch05] can thus be enhanced by a refinement controller. This controller restricts refinement to those (semantical) units which refinement is context- and side-effect free. It is based on operations supporting development projection, renaming, parallel composition of ASM, concatenation and sequential composition of ASM. These refinement operations support common tasks used in specification such as scoping, instantiation and composition of ASM. According to [Bör03] refinement $\text{Ref} = ((\text{RefOp}), (\sigma_\alpha_1(V_1), \sigma_\alpha_2(V_2), \text{corr}), ((\sigma_\alpha_1(D_1), \sigma_\alpha_2(D_2), \equiv)), (\text{Runs}_1, \text{Runs}_2))$ is specified by a notion of refinement from one ASM to another, a scopes on the vocabulary of the two ASM and a notion of correspondence between these two vocabularies, by states of interest and a notion of equivalence within the scopes, and by computational segments of the two ASM. The refinement framework adds constraints for the application of refinement operations $\text{RefOp}$ and context that is affected by the refinement operation beyond the scope on the vocabularies. The ASM method starts the design process at a high level of abstraction. It improves control of development and facilitates verification and synthesis. We thus can link properties of lower levels from analysis at higher level. This development process results in development by refinement. We thus choose the most appropriate level of abstraction for the particular development task. This development methodology allows to relate features of a system with each other. Some of them may be organised as hierarchy of features based on refinement relationships.

4.3 Generic Refinement Steps and Their Correctness

[Bör03, Sch05] have developed a general theory to refinement. Control of correctness of refinement takes into account (a) a notion of refined state and refined vocabulary, (b) a restriction to states of interest, (c) abstract computation segments, (d) a description of locations of interest, and (e) an equivalence relation among those states of interest. The theory developed in [Bör03, Sch05] allows to check whether a given refinement is correct or not.

A typical engineering approach to development of work products such as programs or specifications is based on a general methodology, operations for specification evolution, and a specification of restrictions to the modelling itself. Each evolution step must either be correct according to some correctness criterion or must lead to obligations that can be used for later correction of the specification. The correctness of a refinement step is defined in terms of two given ASM together with the equivalence relations. Already in [Sch05] it has observed that refinement steps can be governed by contracts. We may consider a number of governments [BST06] in the sense of [Cho82]. However we should take into account the choices for style and perspectives.

Given a refinement pattern, perspectives, styles and contract, we may derive generic refinement steps such as data refinement, purely incremental refinement, submachine refinement, and $(m,n)$ refinement. The generic refinement is adapted to the assumptions
made for the given application and to consistency conditions. Typically such consistency are binding conditions of rules to state and vocabulary through the scope of rules. The general approach we envision is depicted in Figure 5.

We are currently developing a number of refinement steps that take preconditions for their enactment and use postconditions for their deployment. The derivation of pre- and postconditions and of is based on principles used for government and binding.

References


Remark: This research proposal is an answer to an email exchange between Daniel Klünder and Andreas Prinz who summarised: “Engineering or modelling of ASM itself has not yet given the right attention.” This paper attempts in development of a general ASM modelling approach.

Acknowledgement: We are very thankful to our reviewers and S. Hegner for their detailed, fruitful and challenging proposals, remarks and requests.
The Conceptual Framework To User-Oriented Content Management

Bernhard Thalheim
Christian-Albrechts-University Kiel, Computer Science Institute, 24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de

Abstract

Content and content management have become buzzwords. They are still heavily overloaded, not well understood or defined and heavily misused. Moreover, the user dimension is not yet incorporated. We develop an approach that is based on separation of concern: syntax dimension and content, semantics dimension and concepts, pragmatics dimension and topics, and finally referent or user dimension and memes. This separation of concern may increase the complexity of handling. We show, however, that a sophisticated handling of different kinds of data at each dimension and a mapping facility between the dimensions provides a basis for a user-oriented content management system. This separation of concern and the special mapping procedure allows to derive content management systems that satisfy the needs of user communities.

1 Web Content Management

Content management, simply stated, is the process of sharing information vital to an organization. Likewise, intranet content management involves sharing information using the private computer networks and associated software of intranets (or extranets) as a primary communication tool [Boi01, SS03]. In today’s “information society,” where the total quantity of data and the pace of communication continue to increase, the goal of effective content management continues to gain importance.

Content management became vital within the web information systems context. A wide variety of systems claim to be a web content management system (CMS), e.g., CacheWare, ConnectSite.com ASP, ContentPlanner, Coremedia, Corevue, Documentum, DynaBase, E-Grail web management platform Ektron, eKeeper, ECOMS, Eprise, Gauss, Imparto Web Marketing Suite, Intervwowren, IntraNet Solutions, iDB Browsinform, iMakeNews.com, Midgard, NCompass, OnDisplay, SiteC, SiteDriver, SiteGeneral, SiteManager, SiteMerger, SiteStation, Stage2Live, Vignette, Website ASP, etc. There are surveys [Jou05] that keep lists of CMS. Roughly we may classify CMS into website CMS, enterprise CMS, advanced document management systems, and extranet CMS. This large variety of systems has a number of properties in common: generation, delivery and storage of complex structured objects; rights management; service management in distributed environment; customer management; update and quality management; context dependent delivery depending on the user, the HCI, and the actual systems situation.
The content of a CMS is a most value asset. Content must be updated frequently to keep user coming back and to succeed in their tasks. Thus, a content management system supports production of content while automating some of the frequent operational tasks.

CMS and web CMS specifically support a variety of tasks:

**Managing web assets:** Content comes from a variety of sources including both file assets, database assets, assets from legacy systems or from syndication services. Content may be stored in both XML and databases. CMS can automate meta data creation and storage which enables companies to organize content and improve customer searches.

**Workflow:** Most CMS provide a user interface for managing tasks such as email notification and approval. Tasks can be manually initiated or automated. Changes are tracked and their history is stored.

**Templates:** Templates are designed for either entering content or for presentation. Templates may contain templates.

**Source control and versioning:** Since data and the generated content changes and older content may be still in use, CMS also provide source code management capabilities such as versioning, merging changes, and identifying conflict resolution.

**Deployment and delivery services:** CMS offer content deployment solutions, automated archival and expiration services, runtime delivery services, and performance improvement tools based on caching approaches.

**Management of distribution and adaptation:** Content is extracted from several sources, is integrated and may be delivered to a large variety of customers.

Therefore, we claim that CMS must

- integrate extraction, preparation, transformation, storage/load and delivery of complex structured objects,
- support workflows and tasks,
- be based on service systems, and
- deliver content objects to users on demand and profile, at the right moment, and within the right format and size.

Content is complex and ready-to-use information. CMS are information systems that support extraction, storage and delivery of complex information. Thus, we claim that content specification must use specification of structuring, functionality, distribution, and interactivity. The co-design approach [Tha00, Tha03] presented in this paper may be used for specification of content structure and of content workflow.

This broad list of requirements, targets, dreams for content management has not yet been supported by any implementation and may lead into the same dead end as high-targeting AI research. This paper shows that a sophisticated separation of concern allows to develop a flexible, powerful and completely satisfying content management. We separate four dimensions: the content dimension for data, the concept dimension for theories and semantics, the
topic dimensions for annotation and referencing, and the referent dimension for handling the concerns of users.

The paper introduces first the first three dimensions, adds in Section 3 the referent dimension and discusses the requirements to advanced CMS that handle the user dimension. Section 4 discusses how to derive the functionality necessary for the development of sophisticated user-oriented CMS and sketch functionality and architecture of advanced CMS.

2 Separating Content, Concepts and Topics

The broad variety of definitions of CMS and the disagreement on a common definition requires to briefly introduce our understanding of CMS and content systems. It is based on the requirement that a content management system must be backed by an information system. A content system [Tha04b] consists of a content management system and a set of content object suites. Content objects may be structured, semi-structured, or unstructured. A suite consists of a set of elements, an integration or association schema [Tha04c] and obligations requiring maintenance of the association[Tha00, Tha03]. In the case of a content suite, we specify content objects based on a type system enabling in describing structuring and functionality of content object, in describing their associations through relationship types and constraints. The functionality of content object is specified by a retrieval expression, the maintenance policy and a set of functions supporting the utilization of the content object and the content object suite.

Content is often considered to be a generalization of knowledge, information, and data. This generalization must capture all aspects of concern. Instead we prefer a separation of aspects of concern:

Pragmatics concentrates on the meaning of terms used by a user.

Semantics expresses the interpretation of terms used by a community of users.

Syntax restricts attention to the language, its construction, and the way of using it through utterances.

This separation is expressed in the semiotic triangle in Figure 1. Content objects are associated with concepts that specify the semantical meaning of content object suites and topics that specify the pragmatical understanding of users. The general association frame is shown in Figure 1. The underlying theories are either theories based on information systems, or on mathematical logics and on concept theory, or on semiotics and corresponding logical theories.

The content-topic pairs are called assets [SS03]. The concept-topic terms are called infons [AFFT05]. Logics calls concept-content pairs semantical units. These pairs may be considered as relations or mappings in Figure 2 such as

interpretation that maps concepts to content suites,

foundation that provides concepts for given content suites,

explanation that maps topics to concepts,

presentation that relates topic suites to content suites,

annotation that represents content suites by topics, and
Content delivery that provides content suites for given topic suites.

Content suites are presented by possible databases, i.e. the data world. The representations may very depending on the model used. Concept worlds are represented by theory worlds. The modeling world depends on the logical theory used for the representation. Topics are used to represent the user world. Topic suites may be represented by ontologies, taxonomies, dictionaries, or glossaries. They are used for communication among users. Therefore, topics are based on a vocabulary a users group has agreed upon. Content management is based on a database and computation environment. Concept management is based on model theory. Topic management uses a presentation, visualization, and language environment.

The functionality necessary for each dimension is based on engines that have been developed in the past:

- **Database and data warehouse system** which handle basic data, derived complex data, extract, transform, and load (ETL) data from one database system to the other one,

- **AI and theorem proving systems** that enable in deriving new pieces of concepts and that support handling of small logical theories, and
The mappings interpretation, foundation, explanation, presentation, annotation, and (content) delivery can be developed using classical Discrete Mathematics or database theory [Tha00]. The concept-content query facility in [TV02] shows that delivery can be based on the product of explanation and interpretation. The association between content and concepts can be defined through queries added to each concept triple \( C \) [FT04]

\[
\text{content schema} \text{ defined on a database schema } S \text{ and query } q \text{ defining the content depending on a database state. Topics } T \text{ are described by the triple}
\]

\[
\text{(user community, topic description, topic population)}
\]

for a given user community (or cultural context based on a population that serves as typical examples for the given topic. Topics are given by an ortho-normalized language [OS96], a glossary, a thesaurus, or an ontology. A glossary is a collection of textual glosses or of specialized terms with their meanings. A thesaurus\(^1\) is a list of subject headings or descriptors about a particular field together with their synonyms usually with a cross-reference system for use in the organization of a collection of documents for reference and retrieval. The word ‘ontology’ is heavily overloaded in the computer engineering area and, thus, not used here.

The annotation may be similarly to [TV02] defined through the product of foundation and presentation. This kind of derived definition of the mapping provides a content independence since the concepts need not to be changed whenever the underlying database or content base is going to be changed.

3 The Referent or User Dimension Extending The Semantic Triangle

Users do not mainly base their utterances on glossaries, thesauri, or ortho-normalized languages. Instead they assume that they will be understood on the basis of context, especially cultural context, their habits, their association to communities or their task background. We may use this association for the development of a user dimensions of advanced CMS. An advanced CMS may be based on the content-concept-topic triangle that uses explicit mappings from the user dimension to this triangle. We explore this idea in the next two sections.

3.1 The Referent or User Dimension for CMS

The Referent Model Language (RML) is the basis for our model for the user dimension of advanced CMS. RML was originally developed in order to support work in heterogeneous databases and data warehousing [Sol98]. RML is based on set theory. Our model is based on set and graph theory. The basic constructs of RML are referent sets and individuals, their properties and relations. These corresponds to the need for expressing interpretations in terms of real-world things. From the area of semantic data models, one has identified a set of general abstraction mechanisms: Classification, aggregation, generalization and association, which are all supported by the language.

Notions of users are memes that are specified by

---

\(^1\)A typical thesaurus is the dictionary developed by wikipedia community groups. The entries in wikipedia are agreed within a certain community but neither validated nor integrated into a common theory.
• names,
• a number of properties,
• a variety of associations with different adhesions to other memes,
• and a variety of groupings for different purposes.

This notion generalizes the notion of knowledge objects developed for knowledge maps. Memes are related to their users. Memes are discussed in [Bla99]. Memes are the units of cultural evolution and selection. They can be folded and be used for derivations. The main operations on memes are understanding, enrichment, and expression. These three kinds of operations are similar to the main database operations: read, compute, and write. In Figure 3 we extend the semantic triangle by the user dimension.

![Figure 3: Extending the semiotic triangle to a tetrahedron for CMS by the referent or user dimension](image)

The large variety of users, their understanding of the world, their slang or “common speaks” make modeling of the referent or user dimension overly complex. We may, however base the understanding of the user dimension on their actions, i.e. what a user (who) intends (purpose, why) to do (how, when) with which content (syntax, what) under which scope (semantics) within which community (pragmatics) with which activities (how, in which order), and in which environment (where). This characterization directly leads to the Zachman modeling framework. The user may

**view** certain content, i.e. sets of basic or derived data defined over an ER schema that has been extended by a set of views,

**express** his/her understanding through **utterances**, i.e. map meme suites to topic suites that are parts of the topic landscapes, and

**chunk** concepts by selecting most appropriate concepts for a given suite of memes.

The semiotic triangle has been extended to a tetrahedron in Figure 3. We may now view this tetrahedron from the user point in Figure 4 based on tripods. The user bases his understanding on content, concepts, and topics on the basis of views, chunks, and utterances, respectively. We additionally consider the data schema and the data necessary for describing content, concepts, topics, and memes. The left tripod in Figure 4 describes the schemata used for specification of the content, concept, topic and meme worlds. The right tripod shows the corresponding data suites used for each layer of concern: content ER data, logical theories, topic landscapes, and user memes.
3.2 Utilization of User Profiles For Advanced CMS

The vast variety of users requires clustering or categorization of users. If the number of categories becomes small then user modeling becomes feasible. In our internet portal projects (e.g., city portals such as www.cottbus.de) we used categorization on the basis of profiles and portfolio. User modeling must be an integral part of any user-oriented CMS. The variety of users may be very high and the task of user user modeling may become infeasible. Defining topics we already used the notion of a user community. This term may be rather broad. Therefore, we integrate the referent or user dimension into advanced CMS based on user profiles and user portfolio.

The user characterization may be rather complex.

If user characterization is, however, based on scales then the user characterization space forms an n-ary cube. The preferences can be then modeled by intervals or spectra. This user preference space may be expressed through Kiviat graphs displayed in Figure 5. The area within the first and second border displays then the user preferences.

User profiles characterize users by user preferences such as

preferences for input devices described on the basis of handling of input types, preferences of specific input types guidance and help during input, control commands, and understanding the input task;

preferences for output devices specified through understanding the type of the output, preferences in specific output types, guidance, help and explanation of the output, control commands, and abilities to understand the output;

The profiles may range from pupils or pensioners interested generally in something to well-informed, educated, critical users seeking additional well-specifiable information. The portfolio may range from inhabitant through tourists to business people seeking special information for their current tasks.
preferences for dialogues such as dialogue properties, dialogue forms and styles, dialogue structuring, dialogue control, and dialogue support necessary;

properties of the user or the user group, e.g., status of the user, formal properties, context of the user, psychological profile, user background and personality factors, training and education, behavioral pattern, need in guidance, and type of the user;

capabilities of the user for task solutions such as understanding the problem area, reasoning capabilities on analogy, realizing variations of the problem solution, solving and handling problems, communication abilities, abilities for explaining results and solutions, and abilities for integration of partial problem solutions;

knowledge of the user, e.g., application knowledge depending on application type, application domain, application structuring, and application functions;

task knowledge, especially task expertise and task experience;

system knowledge depending on the systems to be explored and used.

This user characterization seems to be very complex at the first glance. We may, however, restrict our specification of user characteristics to linearly ordered domain types. For instance, type of users may be ‘casual user’, ‘novice user’, ‘knowledgeable intermittent user’ and ‘expert user’ thus forming a scale.

We extend for this purpose the view definition by parameters that provide the flexibility for meeting the user characteristics. In the next Section we explore how this facility may be implemented.

3.3 Natural Language Foundation For User Portfolio

A portfolio consists of

- a specification of tasks,
- a specification of the context [ST05] of the actor,
- a specification of rights, prohibition, and obligations,
- a specification of the role of actors\(^3\), and
- execution models for fulfilling the requirements including priorities and time and resource restrictions.

The task is given by [Pae00]

- a specification of current and target states,
- a characterization of goals of the task,
- a number of operations that might be used to achieve the task,

\(^3\)Actors are abstractions of groups of users that have the same intention and share goals.
• a metrics for evaluation of distance from the target state and the progress of completion,
• a characterization of knowledge necessary for completing the task, and
• a set of control frames characteristically used for completion of the task.

The workflow of a task completion may be specified through UML activity diagrams or Site-Lang scenario [ST05].

Now the portfolio for users can be given by a set of parameterized views. Using this facility we meet the requirements of users in a flexible form. The next section is, thus, devoted to the conceptual development of the framework for the mapping functions.

User express their questions, their update requirements, and their input or deletion on the basis of natural language utterances relating memes to topics, their understanding of chunks of logical theories and their views on the content data. The development of user functionality may be based on the narrative expressibility of users. This expressibility is based on natural languages. In Indo-European languages verbs express activities. Activities of users may be characterized by verbs of action [Hau00] such as buy, learn, and inform, ergative verbs such as escape, process verbs such as fall asleep (ingressive verbs) and wither (regressive processes) and verbs describing a state such as sleep or have.

For modeling activities of users of advanced CMS we are concentrating on the first and last groups. Within these groups we distinguish with [Kun92]

(1) verbs describing what takes place,
(2) verbs of increasing properties of states,
(3) verbs of coincidence/differentiation,
(4) verbs of communication,
(5) verbs of argumentation,
(6) verbs of agreement,
(7) verbs of chairing,
(8) verbs of collaboration,
(9) verbs of sensuous observation,
(10) verbs of nutrition, and
(11) verbs of cleaning.

The first eight groups are relevant for CMS and may be used for functionality development. The functionality of advanced CMS may be based on discourse types known from conversation theory:

Actions: The partner is requested to do something.

Clarification: The semantics of a partial topic map is becoming specialized and derived.

Decision: The partners agree on next steps to be taken.

Orientation: An orientation for the next actions of the partner is provided.
We can, thus specify a CMS portfolio of the user by and algebraic expression of SiteLang with the basic portfolio elements given by

**Tasks** to be completed by the user,

**Context** of the user within the portfolio,

**Rights, obligations, and prohibitions** for the given step,

**Discourse types** such as action, clarification, decision, or orientation,

**Execution model** to be applied for the user step.

### 3.4 Handling The Vast Variety of Usage Invocations

Modeling of the referent dimension is currently considered one of the most difficult tasks or often considered to be infeasible. The complex behavior of the user may be modeled through the story space [ST05] that describes portfolio under consideration and supports adaptation to users. Based on the approach developed in this section we are able to overcome this problem by

- collecting the actual profiles and portfolio of the current user depending on the actual usage,
- integrating the actual usage into the usage star that consists of a combined profile and a suite of interrelated portfolios, and
- assembling the topic landscape based on usage stars by associating the user memes to those topics that correspond to user communities which work on tasks that are related to tasks within the usage star and that are supporting users of the profile that is valid for the current user and collected within the usage star.

Given a user $u$. The user $u$ has a profile or a number of profiles which can be combined through nesting\(^4\) into $CurrProfile(u)$.

Furthermore, given a set $CurrPortfolio(u)$ of current portfolio of this user. We may now use the star type $UsageStar(u)$ [Tha04a] that combine the common properties and tasks of the user $u$ and the portfolios.

For instance, a user Thalheim that currently works on conference paper evaluation of papers $p_1, ..., p_k$ (decision), seeks for information on authors $a_1, ..., a_l$ (clarification), requests for papers on topics $t_1, ..., t_m$ (orientation), compares the paper results with results $r_1, ..., r_n$ (orientation), uses an email system (actions) etc. and uses the profile of an informed Linux user in a high speed environment. This usage star may be now associated to the combined topic landscape that contains $t_1, ..., t_m$ together with their related topics of distance less than 3, the search interfaces of engines accessible in his current environment or paid on the basis of his profile. The topic landscape is explained through concepts $c_1, ..., c_o$ and associated with the content objects for paper evaluation, with the content objects that are related to the papers of the authors or on the topics of interest.

\(^4\)Depending on the profile specification we may assume that the current profiles of a user are given by a set of ER objects that may be combined into a complex nested object. The operations developed for advanced ER models (e.g., join, product, unnest, nest, rename, difference, set operations [Tha00]) may be used to define the profile combination operations.
3.5 Approaches for Coping With User Understanding

Classically, reasoning of users is associated with deduction that is based on first-order predicate logics. This approach is far too strict. For this reason we develop a more flexible approach to user reasoning. Reasoning of users can be characterized by their specific abilities to relate memes to each other. Reasoning might depend on the knowledge, experience, capabilities of users to reason.

So, the first step consists of the development of an adequate logics that may be different of the one of classical logics usually forced to be used:

- Users use denotations for representing their observations and belief on the reality. These denotations can be mapped to variables. The signification (intension and comprehension) and the meaning (reference, étendue) of these variables may vary depending on the user world and the user memes we are considering.

- The logical connectives $\neg$, $\land$, $\lor$ and the quantifiers $\forall$, $\forall_{\text{time} \text{context}}$, $\exists$ and their logical consequences (e.g., $\alpha \land \beta \models \beta \lor \gamma$) may be different depending on the scope of the user world and their memes.

- Identity, existence and identification vary in users world.

- Classical predicates such as $<$, $>$, $=$, $\leq$, $\geq$ may neither be complete nor transitive. The predicate $\neq$ may be transitive or anti-symmetric.

- Implication may be understood in a large variety. We may distinguish between material implication, weak implication, strong implication, and logical implication.

- Reasoning of users may be based on closed-world or open-world assumptions.

- User may use qualitative reasoning instead of logical reasoning.

- Compositionality of connectives may only be partially accepted. We cannot assume in general validity of $\{\alpha, \beta\} \models \alpha \land \beta$.

- Users, user schemata and user memes may be represented in many-dimensional spaces. For instance, users may use some understanding of space and time. In some cases these dimensions can be modeled by geometric or topological structures.

- Understanding and reasoning of users is context-dependent. Applications often require adaptation of processing context, e.g. to actual environments such as client, server, and channel currently in use, to users rights, roles, obligations, and prohibitions, to content required for the current portfolio for the current user, to actual user with preferences, to level of task completion depending on the user, and to users completion history.

- Utterance of users may be recursively constructed. User may use metaphors and other rhetorical figures which meaning cannot be reconstructed based on the structure of the utterance.

- Usage of memes may depend on the context, on the auditory, on the purpose and other environmental parameters.
This variety may be considered to be the playground of logicians.

At the same time, users may base their reasoning on a variety of approaches. Classically, main logical reasoning procedures are based on the three main reasoning facilities developed for logics:

**Exact reasoning by deduction** uses derivation rules such as the modus ponens

\[ \forall x (P(x) \implies Q(x)), P(a) \quad \frac{Q(a)}{P(a)} \]

for forward deduction and derivation of new formulas or for backward deduction, i.e. tracking back from the proof goal to axioms.

**Reasoning based on induction** uses a background theory \( B \) and observational data \( D \) with the limitation \( B \not\models D \). It is based on such for a formula \( \alpha \) that is consistent with the data \( (B \cup D \not\models \neg \alpha) \) and explains the data \( (B \cup \{\alpha\} \models D) \).

**Abductive reasoning** allow to derive explanations \( E \) within a set of hypotheses \( H (E \subseteq H) \) for observations \( O \) on the basis of a logical theory \( \Sigma \), i.e. we seek for a set \( E \) through which a user may explain the observations \( \Sigma \cup E \models O \). We may require that the set \( \Sigma \cup E \) is consistent. Rules such as the pseudo modus ponens \( \forall x (P(x) \implies Q(x)), Q(a), P(a) \quad \frac{\alpha \implies \beta, \beta \implies \alpha}{\alpha} \) or the modus tollens \( \frac{\alpha \implies \beta, \beta \implies \alpha}{\alpha} \) may be used.

In reality, however, users base their reasoning on other approaches:

**Non-monotonous reasoning** supports reconsideration and revision of conclusions drawn before whenever observations are changing or the belief of the user is under change. In some case the change is only applied depending on the context of the utterance currently under consideration.

**Approximative reasoning** is used whenever fuzzy, uncertain, or unsafe statement, aggregations or conclusions, or their combinations or accumulations are used. We may map such reasoning facilities to point-wise reasoning based on certainty factor methods, Bayes, or many-valued logics, to interval-based logics such as Dempster-Shafer logics or to distribution-based logics such as the logic of possibilities or plausibility logics.

**Temporal reasoning** of users is based on their understanding of modality and time.

**Epistemic reasoning** allows to bind the user understanding to the current user and to handle at the same time reasoning facilities of groups of users.

**Qualitative reasoning** supports the utilization of abstractions and reasoning for abstractions.

At the same time, users are used to partial inconsistencies. The classical approach is to use para-consistent logics. We prefer to extend the theory of knowledge islands [BC95].

The extension is based on quasi-classical logics [BH95]. They support derivation of conclusions in the context of inconsistencies. They use the reasoning facilities sketched above and additionally natural deduction based on the Gentzen calculus. This logics support the unambiguous identification of each derived formula. This identification is compositional, i.e. two derived formulas are identified by the union of their identifications. So, the user sees the effect or impact of the conclusions drawn. A knowledge island of a user is a maximal consistent set of users memes. Users may use a number of knowledge islands at the same time. Conclusions are only drawn within the knowledge island.

At the same time, we characterize the languages we use for representation of memes and for reasoning based on memes by different layers of adequacy:
Epistemic adequacy characterizes the expressive strength of the language used.

Heuristic adequacy uses a complexity characterization for checking whether a derivation procedure is feasible and can be applied or whether it should not be applied.

Ergonomic adequacy considers whether a user can easily understand the reasoning facilities and their results.

Cognitive adequacy associates derivations with the users ability to understand the conclusions drawn.

Users reasoning abilities are characterized by their logical language that is used for representation of memes, for associating memes by connectives and quantifiers, and for constructing formulas on memes, reasoning procedures such as combined inductive reasoning and qualitative reasoning, and their abilities to cope with inconsistencies on the basis of knowledge islands.

4 Development of Systems Supporting User-Oriented Content Management

User-oriented CMS are not yet established and developed. We derive now a number of general properties of such systems and an architecture for such systems.

4.1 Faithful, Consistent And Well-Founded User-Oriented CMS

Properties introduced above may be now used to define properties of the user-oriented CMS:

Well-foundedness: A CMS is well-founded if the two subset properties

\[(\text{interpretation(}t\text{)}} \subseteq \text{delivery(}t\text{)}} \quad \text{and} \quad \text{presentation(}cs\text{)}} \subseteq \text{annotation(}cs\text{)}}

are valid for any topic \(t\) and any content suite \(cs\).

Faithfulness: A user-oriented CMS is faithful if

\[(\text{interpretation(} \text{explanation}(\text{associate}(m))) \subseteq \text{delivery}(\text{associate}(m)))

for any meme \(m\).

Saturatedness: A CMS is saturated if

\[(\text{interpretation(}t\text{)}} \supseteq \text{delivery(}t\text{)}} \quad \text{and} \quad \text{presentation(}cs\text{)}} \supseteq \text{annotation(}cs\text{)}}

are valid for any topic \(t\) and any content suite \(cs\).

Consistency: A CMS is consistent if

\[(\text{interpretation(} \text{explanation}(\text{associate}(m))) \supseteq \text{delivery}(\text{associate}(m)))

for any meme \(m\).

Based on these properties we need to solve the following problems:
Foundation problem: A CMS is well-founded if no topic exists that may be associated with a concept or a concept set which are associated to content data which are not annotated by the given topic. So, the foundation problem consists in association of all topics which are not well-founded.

Saturation problem: If all topics that are associated to content data that are founded for this topic then the system is saturated. We need now to find an efficient procedure for correction.

Faithfulness problem: The system becomes faithful if all memes of users are represented by faithful topics. So, the problem consists in finding those memes which do not have an association to founded topics.

Consistency problem: We need to detect those memes that are not associated to saturated topics and then to repair this inconsistency.

Profile genericity problem: Profiles of users can be ordered by their level of abstraction. The problem whether there exists a small set of abstract profiles that can be specialized to the specific ones may be solved if the user domain is homogeneous.

Profile initialization problem: User profiles may be initially specified by some initial profiles, e.g. Faithful PC member, Late PC member. The problem is to find a sufficient large set of initial profiles.

Profile extension problem: Profiles are easy to manage if the profile set can be hierarchically ordered. The problem whether we can find a hierarchically ordered set of profiles and then consider any profile extension through moving from a less detailed profile to a more detailed one may be solved if the variety of profiles is small or restricted by the application domain.

Portfolio genericity problem: Portfolio may be ordered by their abstractness. We need to find such a set of abstract portfolio that can be refined or specialized to more specific ones.

Portfolio initialization problem: The specification of tasks may be given first based on very general descriptions similar to the generality order of words in natural languages. We need to solve whether there exists a small set of very general portfolio that can be used as main initial portfolio.

Portfolio extension problem: The consistent and faithful extension of portfolio seems to achievable if the portfolio can be extended with full knowledge of the impact and consequences of this extension.

This set of problems may be considered as open problems. Similar to [Tha04a] we may however base profiles and portfolio on multidimensional characterizations that can easily be combined.
4.2 Functions Mapping Between Memes And Concepts, Content, and Topics

The three additional structures beside infons, assets and units are chunks associating concepts with memes, utterances associating topics with memes, and views associating content with memes.

We must now develop an architecture for mapping content, concepts, topics and memes to each other. These mapping must be based on existing technology. Before providing a technological framework we discuss the variety of mappings. At the same time, the mappings must preserve consistency and must provide a basis for development of facilities for user communities.

We must now consider a number of different views:

- The user understands chunks of concepts.
- The user expresses data needs through utterances based on association to topics.
- The user queries for content or data through views.

This variety may be managed in a simpler form if we use well-founded and saturated CMS. In this case, natural layering uses four layers of data, content, concepts, and topics displayed in Figure 6.

<table>
<thead>
<tr>
<th>Layer 4: Memes of the users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 3: Topics of the topic landscapes for annotation and representation</td>
</tr>
<tr>
<td>Layer 2: Concepts of the concept bases for foundation and explanation</td>
</tr>
<tr>
<td>Layer 1: Content of the content bases as macro-data or aggregations</td>
</tr>
<tr>
<td>Layer 0: Data and documents of the underlying databases as micro-data</td>
</tr>
</tbody>
</table>

Figure 6: The data layers of well-founded and saturated CMS

Functionality of the well-founded and saturated CMS is simply based on the mappings interpretation and explanation and their ‘inverses’ presentation and foundation. We may have a high initial effort for building such systems and a substantial update effort. We may, however, use a ‘liberal’ approach that is based on lazy foundation and lazy saturation. In this case, we generate a number of correction tasks. Programming of such correcting facilities can easily be based on the throw, try-catch facilities of languages such as Java.

The main facilities of the top-layer of user-oriented CMS are, thus:

The utterance interpreter and analyzer support the analysis of utterances made by the user and the generation of the appropriate topic landscape for an utterance or a set of utterances.

The portfolio manager allows to derive, to manage, to change, to retrieve and to associate portfolio of the user. The portfolio manager may use a specific task glossary that supports analysis of the meaning of utterances.
The **profile manager** supports storage, retrieval, change, and introduction of user profiles. User profiles may include specific slang-like vocabularies.

The **meme manager** supports the storage, manipulation, and retrieval of memes.

The systems necessary for the management, interpretation and retrieval of utterances, memes, profiles, and portfolio are rather classical systems. The utterance interpreter and analyzer may use the ER NL-modeling tools [Tha00] and the theory developed in [Hau00]. Portfolio, profile and meme manager are specific database systems that handle profiles, portfolio, and memes. In this case, the development of the database structures representing profiles, portfolio, and memes is the most important problem solving step. Therefore, we are sure that the proposed framework may be the basis for user-oriented CMS.

4.3 **Proposing An Architecture Of A User-Oriented CMS**

The broad variety of definitions of CMS and the disagreement on a common definition requires to briefly introduce our understanding of content management systems and content systems. It is based on the requirement that a content management system must be backed by an information system.

We may envision the general architecture of a user-oriented CMS. It consists of a content management system that uses a web playout system as shown in Figure 7. The architecture is based on the proposal of [FT04] for content management systems and the proposal of [ST05] for web information systems. The first proposal used the 2-layer architecture of content management that are defined over database systems by adding content services with content structuring and content functionality. A content management system, thus, consists of an information system extended by facilities for management of content suites. The second proposal has defined web information systems through a playout facility with containers for adapted content delivery to the web playout system that uses an explicit specification of the story space. Our new proposal generalizes an architecture to information system that has successfully been applied in more than 30 projects resulting in huge or very large information-intensive websites and in more than 100 projects aiming in building large information systems.

This CMS is now extended by a concept management system that supports reasoning on concepts and management of infons and units. The topic management system is an extension of the system discussed in [TV02] and supports infons and assets. The user management system supports user adaptation, user management, profile and portfolio management.

5 **Conclusion**

This paper does not target to develop the ultimate solution for all user-oriented CMS. We developed a framework that allows to manage user-oriented CMS by

- **separating concerns in dimensions** for data, logical foundations and representational (topic) worlds,
- **handling each of the dimensions separately** by providing sophisticated functionality for the dimension,
- **adding the user worlds** through explicit representation of their understandings, and
mapping facilities between the syntax, semantics, pragmatics and referent dimension.

This framework has already been partially used in our web information systems projects. The project DigiCult (www.museen-sh.de) that already contains a CMS, a web playout system and a topic management system is currently extended by a user management system. Within this project we are now experimenting with the proposed framework to user-oriented CMS.

The proposed separation between the syntactical and semantical dimensions has led to the integration of sophisticated derivational facilities into classical content management systems. The separation between the syntactical and pragmatical dimension has already intensionally been used in a number of commercial CMS. The separation between the semantical and pragmatical dimensions intensionally led on the basis of AI research. The integration of the referent dimension was a dream over decades for database and information systems development. Despite the Scandinavian school of conceptual modeling and a number of Japanese groups working within the 5th generation project and in the Meme Media Laboratory of Hokkaido University, the user dimension has been neglected in research. This paper provides a uniform and feasible framework for user-oriented content management.

References


Remark: Our main aim has been the development of a general theory of content management. We used the co-design framework [Tha03] to content management. We restrict thus the bibliography only to those references which are necessary for this paper. An extensive bibliography on relevant literature in this field can be found in [Tha00].
Engineering Database Component Ware

Bernhard Thalheim

Christian Albrechts University Kiel, Department of Computer Science, 24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de

Abstract. Large database applications often have a very complex structuring that complicate maintenance, extension, querying, programming. Due to this complexity systems become unmaintainable. We observe, however, that large database applications often use an implicit structuring into connected components. We propose to initially use this internal structuring for application development. The application architecture is based on database components. Database components can be composed to an application system. This paper shows how components may be developed, composed and applied.

1 Towards Information Systems Engineering

Component-Based Application Engineering

Software engineering is still based on programming in the small although a number of approaches has been proposed for programming in the large. Programming in the large uses strategies for programming, is based on architectures, and constructs software from components which collaborate, are embedded into each other, or are integrated for formation of new systems. Programming constructs are then pattern or high-level programming units and languages.

The next generation of programming observed nowadays is programming in the world within a collaboration of programmers and systems. It uses advanced scripting languages such as Groovy with dynamic integration of components into other components, standardisation of components with guarantees of service qualities, collaboration of components with communication, coordination and cooperation features, distribution of workload, and virtual communities. Therefore, component engineering will also form the kernel engineering technique for programming in the world. The next generation of software engineering envisioned is currently called as programming by composition or construction. In this case components also form the kernel technology for software and hardware.

Software development is mainly based on stepwise development from scratch. Software reuse has been considered but never reached the maturity for application engineering. Database development is also mainly development in the small. Schemes are developed step by step, extended type by type, and normalized locally type by type. Views are still defined type by type although more complex schemata can be easily defined by extended ER schemata [Tha00].

Therefore, database engineering must still be considered as handicraft work which require the skills of an artisan. Engineering in other disciplines has already gained the maturity for industrial development and application.
Engineering applications have been based on the simple separation principle: Separation of elements which are stable from those elements which are not. This separation allows standardization and simple integration. An example is the specification of screws as displayed in Figure 1. Screws have a standardized representation: basic data, data on the material, data on the manufacturing, data on specific properties such as head, etc.

Complex Applications Result in Large Schemata
Monographs and database course books usually base explanations on small or ‘toy’ examples. Reality is, however, completely different. Database schemata tend to be large, not surveyable, incomprehensible and partially inconsistent due to application, the database development life cycle and due to the number of team members involved at different time intervals. Thus, consistent management of the database schema might become a nightmare and may lead to legacy problems. The size of the schemata may be very large.

It is a common observation that large database schemata are error-prone, are difficult to maintain and to extend and are not surveyable. Moreover, development of retrieval and operation facilities requires highest professional skills in abstraction, memorization and programming. Such schemata reach sizes of more than 1000 attribute, entity and relationship types. Since they are not comprehensible any change to the schema is performed by extending the schema and thus making it even more complex. Database designers and programmers are not able to capture the schema.

Application schemata could be simpler only to a certain extent if software engineering approaches are applied. The repetition and redundancy in schemata is also caused by

- different usage of similar types of the schema,
- minor and small differences of the types structure in application views, and
- semantic differences of variants of types.

Therefore, we need approaches which allow to reason on repeating structures inside schemata, on semantic differences and differences in usage of objects.

---

1 We use the extended ER model [Tha00] that allows to display subtypes on the basis of unary relationship types and thus simplifies representation.
Large schemata also suffer from the deficiency of variation detection: The same or similar content is often repeated in a schema without noticing it.

Techniques to Decrease Complexity in Applications
Large database schemata can be drastically simplified if techniques of modular modelling such as modular design by units [Tha00] are used. It is an abstraction technique based on principles of hiding and encapsulation. Design by units allows to consider parts of the schema in a separate fashion. The parts are connected via types which function similar to bridges.

Data warehousing and user views are often based on snowflake or star schemata. The intuition behind such schemata is often hidden. Star and snowflake schemata are easier to understand, to query, to survey and to maintain. At the same time, these structures are of high redundancy and restricted modelling power. For instance, the central type in a star or snowflake schema is a relationship type which has attributes that use only numerical types. We may wonder, however, why we need to apply these restrictions and why we should not use this approach in general.

Co-design [Tha00] of database applications aims in consistent development of all facets of database applications: structuring of the database by schema types and static integrity constraints, behavior modelling by specification of functionality and dynamic integrity constraints and interactivity modelling by assigning views to activities of actors in the corresponding dialogue steps. Co-design, thus, is based on the specification of the the database schema, functions, views and dialogue steps. At the same time, various abstraction layers are separated such as the conceptual layer, requirements acquisition layer and implementation layer.

Software becomes surveyable, extensible and maintainable if a clear separation of concerns and application parts is applied. In this case, a skeleton of the application structure is developed. This skeleton separates parts or services. Parts are connected through interfaces. Based on this architecture, an application can be developed part by part.

We combine modularity, star structuring, co-design, and architecture development to a novel framework based on components. Such combination seems to be not feasible. We discover, however, that we may integrate all these approaches by using a component-based approach. This skeleton can be refined during evolution of the schema. Then, each component is developed step by step. Structuring in component-based co-design is based on two constructs:

Components: Components are the main building blocks. They are used for structuring of the main data. The association among components is based on 'connector' types (called hinge or bridge types) that enable in associating the components in a variable fashion.

Skeleton-based construction: Components are assembled together by application of connector types. These connector types are usually relationship types.

Goals of the Paper
The paper surveys our approach [Tha02, Tha03a, Tha05] for systematic development of large database schemata and applies it for database construction based on components and for collaborating component suites. The paper is based on [Fey03, FT02, ST06a].
We introduce first the concept of database components and then discuss engineering of database applications based on components.

2 Database Components and Construction of Schemes

Database Schemes in a Nutshell
We use the extended ER model for representation of structuring and behavior generalizing the approach of [PBGG89]. The extended ER model (HERM) [Tha00] has a generic algebra and logic, i.e., the algebra of derivable operations and the fragment of (hierarchical) predicate logic may be derived from the HERM algebra whenever the structure of the database is given.

A database type $S = (S, O, \Sigma)$ is given by
- a structure $S$ defined by a type expression defined over the set of basic types $B$, a set of labels $L$ and the constructors product (tuple), set and bag, i.e. an expression defined by the recursive type equality $t = B | t \times \ldots \times t | \{t\} | [t] | l : t$,
- a set of operations defined in the ER algebra and limited to $S$, and
- a set of (static and dynamic) integrity constraints defined in the hierarchical predicate logic with the base predicate $P_S$.

Objects of the database type $S^C$ are $S$-structured. Classes $S^C$ are sets of objects for which the set of static integrity constraints is valid.

Operations can be classified into “retrieval” operations enabling in generating values from the class $S^C$ and “modification” operations allowing to change the objects in the class $S^C$ if static and dynamic integrity constraints are not invalidated.

A database schema $D = (S_1, \ldots, S_m, \Sigma_G)$ is defined by
- a list of different database types and
- a set of global integrity constraints.

The HERM algebra can be used to define (parameterized) views $V = (V, O_V)$ on a schema $D$ via
- an (parameterized) algebraic expression $V$ on $D$ and
- a set of (parameterized) operations of the HERM algebra applicable to $V$.

The view operations may be classified too into retrieval operations $O^R_V$ and modification operations $O^M_V$. Based on this classification we derive an output view $O^V$ of $V$ and an input view $I^V$ of $V$.

In a similar way (but outside the scope of this paper) we may define transactions, interfaces, interactivity, recovery, etc.

Obviously, $I^V$ and $O^V$ are typed based on the type system. Data warehouse design is mainly view design [Tha00].

Database Components and Component Algebra
A database component is database scheme that has an import and an export interface for connecting it to other components by standardized interface techniques. Components are defined in a data warehouse setting. They consist of input elements, output
elements and have a database structuring. Components may be considered as input-output machines that are extended by the set of all states $S_C$ of the database with a set of corresponding input views $I_V^C$ and a set of corresponding output views $O_V^C$. Input and output of components is based on channels $K$. The structuring is specified by $S_K$. The structuring of channels is described by the function $\text{type} : C \rightarrow \mathcal{V}$ for the view schemata $\mathcal{V}$. Views are used for collaboration of components with the environment via data exchange. In general, the input and output sets may be considered as abstract words from $M^*$ or as words on the database structuring.

A database component $K = (S_K, I_V^K, O_V^K, S_C^K, \Delta_K)$ is specified by

- (static) schema $S_K$ describing the database schema of $K$,
- syntactic interface providing names (structures, functions) with parameters and database structure for $S_C^K$ and $I_V^K, O_V^K$,
- behavior relating the $I_V, O_V$ (view) channels

$$\Delta_K : (S_C^K \times (I_V^K \rightarrow M^*)) \rightarrow P(S_C^K \times (O_V^K \rightarrow M*))$$

Components can be associated to each other. The association is restricted to domain-compatible input or output schemata which are free of name conflicts.

Components $K_1 = (S_1, I_V^1, O_V^1, S_C^1, \Delta_1)$ and $K_2 = (S_2, I_V^2, O_V^2, S_C^2, \Delta_2)$ are free of name conflicts if the set of attribute, entity and relationship type names are disjoint. Channels $C_1$ and $C_2$ of components $K_1 = (S_1, I_V^1, O_V^1, S_C^1, \Delta_1)$ and $K_2 = (S_2, I_V^2, O_V^2, S_C^2, \Delta_2)$ are called domain-compatible if

$$\text{dom}(\text{type}(C_1)) = \text{dom}(\text{type}(C_2))$$

An output $O_V^1$ of the component $K_1$ is domain-compatible with an input $I_V^2$ of the component $K_2$ if

$$\text{dom}(\text{type}(O_V^1)) \subseteq \text{dom}(\text{type}(I_V^2))$$

Component operations such as merge, fork, transmission are definable via application of superposition operations [Kud82, Mal70]: Identification of channels, permutation of channels, renaming of channels, introduction of fictitious channels, and parallel composition with feedback displayed in Figure 2.

![Fig. 2. The Composition of Database Components](image)

Thus, a component schema is usually characterized by a kernel entity type used for storing basic data, by a number of dimensions that are usually based on subtypes of the entity type which are used for additional properties. These additional properties are clustered according to their occurrence for the things under consideration. Typically, the component schema uses four dimensions: subtypes, additional characterization, versions and meta-characterizations.
The star schema is the main component schema used for construction. A star schema for a database type $C_0$ is defined by

- the (full) (HERM) schema $S = (C_0, C_1, ..., C_n)$ covering all types on which $C_0$ has been defined,
- the subset of strong types $C_1, ..., C_k$ forming a set of keys $K_1, ..., K_s$ for $C_0$, i.e.,
  \[ \bigcup_{i=1}^{s} K_i = \{C_1, ..., C_k\} \quad \text{and} \quad K_i \rightarrow C_0, \quad C_0 \rightarrow K_i \quad \text{for} \quad 1 \leq i \leq s \]
  and \[ \text{card}(C_0, C_i) = (1, n) \quad \text{for} \quad (1 \leq i \leq k) \]
- the extension types $C_{k+1}, ..., C_m$ satisfying the (general) cardinality constraint \[ \text{card}(C_0, C_j) = (0, 1) \quad \text{for} \quad ((k + 1) \leq i \leq n) \].

The extension types may form their own $(0, 1)$ specialization tree (hierarchical inclusion dependency set). The cardinality constraints for extension types are partial functional dependencies.

There are various variants for representation of a star schema:

- Representation based on an entity type with attributes $C_1, ..., C_k$ and $C_{k+1}, ..., C_l$ and specialisations forming a specialization tree $C_{l+1}, ..., C_n$.
- Representation based on a relationship type $C_0$ with components $C_1, ..., C_k$, with attributes $C_{k+1}, ..., C_l$ and specialisations forming a specialization tree $C_{l+1}, ..., C_n$. In this case, $C_0$ is a pivot element \[ [BP00] \] in the schema.
- Representation by be based on a hybrid form combining the two above.

Star schemata may occur in various variants within the same conceptual schema. Therefore, we need variants of the same schema for integration into the schema. We distinguish the following variants:

Integration and representation variants: For representation and for integration we can define views on the star type schema with the restriction of invariance of identifiability through one of its keys. Views define ‘context’ conditions for usage of elements of the star schema.

Versions: Objects defined on the star schema may be a replaced later by objects that display the actual use, e.g., Documents are obtained and stored in the Archive.

Variants replacing the entire type another through renaming or substitution of elements.

History variants: Temporality can be explicitly recorded by adding a history dimension, i.e., for recording of instantiation, run, usage at present or in the past, and archiving.

Lifespan variants of objects and their properties may be explicitly stored. The lifespan of products in the acquisition process can be based on the Product-Quote-Request-Response-Requisition-Order-InventoryItem-StoredItem cycle displayed in Figure [8].

Meta-Characterization of Components, Units, and Associations

Utilization information is often only kept in log files. Log files are inappropriate if the utilization or historic information must be kept after the data have been changed. Database applications are often keeping track of utilization information based on archives. The same observation can be made for schema evolution. We observed that database schemata change already within the first year of database system exploitation. In this case, the schema information must be kept as well.
The skeleton information is kept by a meta-characterization information that allows
to keep track on the purpose and the usage of the components, units, and associations.
Meta-characterization can be specified on the basis of docketes [SS99] that provide in-
formation. The following frames follows the co-design approach [Tha00] with the inte-
grated design of structuring, functionality, interactivity and context. The frame is struc-
tured into general information provided by the header, application characterization, the
content of the unit and documentation of the implementation.

- on the content (abstracts or summaries),
- on the delivery instruction,
- on the parameters of functions for treatment of the unit (opening with(out) zoom-
ing, breath, size, activation modus for multimedia components etc.)
- on the tight association to other units (versions, releases etc.),
- on the meta-information such as resources, restriction, copyright, roles, distribution
  policy etc.
- on the content providers, content reviewers and review evaluators with quality con-
trol policies,
- on applicable workflows and the current status of completion and
- on the log information that enable in tracing the object’s life cycle.

Dockets can be extended to general descriptions of the utilization. The following def-
inition frame is appropriate which classifies meta-information into mandatory, good
practice, optional and useful information.

3 Non-invasive Database Component Composition

Construction Requirements
Component construction is based on a general component architecture or a skeleton.
Each component is developed in separate. The advantage of the strict separation is an
increase of modularisation, parameterisability and conformance to standards.

We derive now a none-invasive construction approach which does not change com-
ponents used for construction. Due to this restriction we gain a number of properties
such as adaptivity, seamless gluing, extensibility, aspect separation, scalability, and
metamodelling and abstraction.

Components and Harnesses
The construction is based on harnesses and the application skeleton. The skeleton is
a special form of a meta-schema architecture. It consists of a set of components and a
set of harnesses for superposition operations. Harnesses are similar to wiring harnesses
used in electrotechnics. A harness consists of a set of input-output channels that can
be used to combine wrapped components.

Given a sets of components \( R = \{K_1, ..., K_m\} \) and labels \( \mathcal{L} = \{L_1, ..., L_n\} \) with
\( n \geq m \). Given furthermore a total function \( \tau : \mathcal{L} \rightarrow R \) used for assigning roles to
components in harnesses. The triple \((R, \mathcal{L}, \tau)\) is called harness skeleton \( \mathcal{H} \). The arity
of the skeleton is \( n \).
The skeleton is graphically represented by doubly rounded boxes. Components are graphically represented by rounded boxes. The construction may lead to complex components called units.

The example in Figure 3 has been used in one of our projects. Parliamentarians and inhabitants are combined into a component Users. We may use a large variety of positions. A user may use a certain service through some devices. Appointments are based on the usage of services. Tools vary depending on services and on equipment. The final schema contains more than 2,500 attribute, entity, cluster and relationship types. The skeleton of the application is rather simple.

Fig. 3. Skeleton of a Schema for e-Government Service Applications

Harness Filters
Components may be associated in a variety of ways. In the application in Figure 3 the usage of services depends on the properties of parties, the tools they may use, and the services provided. Services, parties, and tools have their own dimensionality. If we use the classical approach to schema development each subtype may cause the introduction of a new usage type. The schema explodes due to the introduction of a large variety of usage type. To overcome this difficulty we introduce filters.

Given component schemata of an n-ary harness skeleton. A filter of an n-ary harness is an n-ary relation defined of the multi-dimensional structure of the components, i.e. on the views defined for the components.

Filters may be represented either graphically or in a tabular form. In our example, we obtain the following filter. Components are already presented in Figure 3. We develop a number of services which might be used depending on the role, rights, and positions of the users. For instance, the parliamentarian is interested in search of related documents in the role of an inhabitant and in search of related meetings.
The implementation of filters is rather straightforward. Each harness has a filter. Since views are defined together with their identification mechanism, an n-ary harness may be represented by an (n + 1)-ary relationship type associating the components with their roles and extended by the filter.

A harness consists of the harness skeleton $\mathcal{H} = (R, L, \tau)$ and the harness filter $\mathcal{F} = \{(L_i, V_{L_i}) | 1 \leq i \leq n, L_i \in L, V_{L_i} \subseteq V_{\tau(L_i)}\}$ for a set of wrapped components $(K_i, V_i)$.

Operators Used For Non-Invasive Schema Construction

In [Tha03b] a number of composition operators for construction of entity and relationship types has been introduced: constructor-based composition, bulk composition, lifespan composition (architecture-based composition, evolution composition, circulation composition, incremental composition, network composition, loop composition), and context composition.

We generalize now these composition operators to component-based schema construction.

**Constructor harnesses** are based on composition operations such as product, nest, disjoint union, difference and set operators.

**Bulk harnesses** allow to bound components, types or classes which share the same skeleton. Two harness skeletons $\mathcal{H}_1 = (R_1, L_1, \tau_1)$ and $\mathcal{H}_2 = (R_2, L_2, \tau_2)$ are called unifiable if they are defined over the same set of components, $|L_1| = |L_2| = n$, and there exists a permutation $\rho$ on $\{1, \ldots, n\}$ such that $K_{\tau_1(i)} = K_{\tau_2(\rho(i))}$. The bulk harness of unifiable harnesses $\mathcal{H}_1, \ldots, \mathcal{H}_p$ is constructed by renaming the labels $L_j$ of each harness $\mathcal{H}_i$ to $L_{i,j}$ and combining the label functions $\tau_i$. 

<table>
<thead>
<tr>
<th></th>
<th>Parl.</th>
<th>Inhab.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sur</td>
<td>Doc</td>
<td>Doc</td>
</tr>
<tr>
<td>Pla</td>
<td>Meet, Doc</td>
<td>Doc</td>
</tr>
<tr>
<td>SRe</td>
<td>Meet</td>
<td>Doc</td>
</tr>
<tr>
<td>Load</td>
<td>Meet</td>
<td>-</td>
</tr>
<tr>
<td>PrCr</td>
<td>-</td>
<td>Doc</td>
</tr>
</tbody>
</table>
**Application-separating harnesses:** An enterprise is usually split into departments or units which run their own applications and use their own data. Sharing of data is provided by specific harnesses.

**Distribution-based harnesses:** Data, functions and control may be distributed. The exchange is provided through specific combinations which might either be based on exchange components that are connected to the sites by harnesses or be based on combination harnesses.

**Application-separation-based harnesses** have been widely used for complex structuring. The architecture of SAP R/3 often has been displayed in the form of a waffle. For this reason, we prefer to call this composition *waffle composition* or *architecture composition* displayed in Figure 4.

![Fig. 4. The Waffle Architecture Composition](image)

An Application of Component Composition

A typical lifespan construction is the *Order* chain displayed in Figure 6. We discover a chain in the ordering and trading process: *Quote, Request, Response, Requisition, Order, Delivery, Billing, Payment*. Within this chain, parameters such as people responsible in certain stages are inherited through the components. They are included into the type for the purpose of simpler maintenance. They cannot be changed within the type inheriting the component. Thus, we use an extended inheritance of structuring beyond the inheritance of identification.

At the same time, this schema can be constructed on the basis of components. We may distinguish only four basic parts. Parties are either organisations or people. Products have a number of properties that are independent on parties. The two components are associated within the ordering and trading process. The parties may play different roles within this process. The parties act based on these roles. So, the component schema is given in Figure 5.

The roles of parties in the ordering and trading process can be unfolded. We observe a role of a supplier, a role of a requestor, a role of a responding party, a role of a requisition party and finally the role of the orderer. At the same time, the final order has a history or a lifespan. We may apply the lifespan constructor as well. The application can be either based on collaborating components are can be condensed to the schema given in Figure 6. This schema combines components and unfolds roles and expands the ordering and trading activities. We notice that this schema is not necessarily the solution for the ordering and
trading process. We may use the components instead and explicitly model component collaboration. In this case the components may stay non-integrated.

4 Collaborating Database Component Suites

Services Provided By Components For Loosely Coupled Suites
A service consists of a wrapped component \((K_i, V_i)\), the competencies \(\Sigma(K_i, V_i)\) provided and properties \(\Psi(K_i, V_i)\) guaranteeing service quality. Wrapped components offer their own data and functions through their views. The competence of a service manifests itself in the set of tasks \(T\) that may be performed and in the guarantees for their quality.

Database Component Collaboration
Instead of expanding and unfolding the component schema in Figure 5, we may follow a different paradigm. The four basic parts are loosely associated by a collaboration, are supported by component databases and communicate for task resolution. This approach has already been tried for distributed databases. Our approach is far more general and provides a satisfying solution.
A collaborating database component suite \( \mathcal{S} = (\mathcal{K}, \mathcal{H}, \mathcal{F}, \Sigma) \) consists of

- an set \( \mathcal{K} \) of wrapped database components \( (K_i, V_i) \)
- a harness consisting of the harness skeleton \( \mathcal{H} = (K, \mathcal{L}, \tau) \) and the harness filter \( \mathcal{F} \),
- an collaboration schema \( \mathcal{F} \) among these components based on the harness, and
- obligations \( \Sigma \) requiring maintenance of the collaboration.

The collaboration schema explicitly models collaboration among components. We distinguish three basic processes of component collaboration:

**Communication** is defined via exchange of messages and information or simply defined via services and protocols [Kon03]. It depends on the choice of media, transmission modes, meta-information, conversation structure and paths, and on the restriction policy. Communication must be based on harnesses.

**Coordination** is specified via management of components, their activities and resources. It rules collaboration. The specification is based on the pre-/post-articulation of tasks and on the description management of tasks, objects, and time. Coordination may be based on loosely or tightly integrated activities, may be enabled, forced, or blocked. Coordination is often specified through contracts and refines coordination policies.

**Cooperation** is the production of work products taking place on a shared space. It can be considered as the workflow or life case perspective. We may use a specification based on storyboard-based interaction that is mapped to (generic and structured) workflows. The information exchange is based on component services [ST06a] for production, manipulation, organization of contributions.

This understanding has become now a folklore model for collaboration but has not yet been defined in an explicit form. We use the separation of concern for the specification of component collaboration.

**Collaboration obligations** are specified through the collaboration style and the collaboration pattern.

The collaboration style is based on four components describing supporting programs of the connected component including collaboration management:

- **data access pattern** for data release through the net, e.g., broadcast or P2P, for sharing of resources either based on transaction, consensus, and recovery models or based on replication with fault management, and for remote access including scheduling of access;
- **the style of collaboration** on the basis of component models which restrict possible communication;
- **and the coordination workflows** describing the interplay among parties, discourse types, name space mappings, and rules for collaboration.

**Collaboration pattern** generalize protocols and their specification [Kon03]. They include the description of components, their responsibilities, roles and rights. We know a number of collaboration pattern supporting access and configuration (wrapper, facade, component configuration, interceptor, extension interface), event processing (reactor, proactor, asynchronous completion token, accept connector), synchronization
(scoped locking, strategized locking, thread-safe interface, double-checked locking optimization) and parallel execution (active object, monitor object, half-sync/half-async, leader/followers, thread-specific storage).

Exchange frames combine the collaboration schema with the collaboration obligations. The collaboration schema can be considered to be an exchange architecture that may also include the workplace of the client using the component suite.

Supporting Collaboration Schemata By Service Managers

The abstraction layer model [Tha00, ST06b] distinguishes between the application domain description, the requirements prescription, the system specification, and the logical or physical coding. The specification layer typically uses schemata for specification. These schemata may be mapped to logical codings. The mapping of services to logical database components is already given by classical database textbooks. We map collaboration schemata to service managers. This mapping provides also a framework for characterisation of competencies and quality.

The service manager \( \text{Man} \) supports functionality and quality of services and manages sets of wrapped components. The manager supports a number of features for collaboration. The architecture of the services manager follow the separation of concern into communication, coordination, and cooperation. We may thus envision the architecture in Figure 7.

![Figure 7. Layers of a services manager for typical collaborating components](image)

Collaborating services are defined by the quadruple \( S = (\mathcal{S}, \text{Man}, \Sigma_S, \Psi_S) \) describing (Collaborating Suite, Service Manager, Competence, Characteristics). The competence is derived from the competence of the services. The quality of collaborating services may also be derived from the quality properties of components in the suite based on the properties of the harnesses, their collaboration schema, and the corresponding obligations. Typically, quality heavily depends on the suite properties. For instance, reliability of a suite may be less than the reliability of its components.

Concluding by Demonstrating the Potential of Privacy Supporting Suites

Let us show the potential of loosely coupled database component suites for privacy workbenches. Privacy research is becoming the “poor cousin” among the mainstream research. Novel applications such as Web2.0 have created a new rush towards social
networking and collaborative applications. This enables new possibilities, but also is a threat to users’ privacy and data. On the surface, many people seem to like giving away their data to others in exchange for building communities or like to get bribes from companies in exchange of privacy. A number of hidden privacy implications of some Web2.0 and Identity2.0 services, standards and applications can be observed here. At the same time, it is often stated that there is no way to properly preserve privacy.

We show the potential of collaborating databases based on the infon model of [AFFT05]. An infon is a discrete item of information of an individual and may be parametric. The parameters are objects, and the so-called anchors assign these objects such as agents to parameters.

We may distinguish four relationships between infons and individuals (people), institutions, agencies, or companies: An infon may be possessed by an individual, institution, agency, or company. For example, an individual may possess private information of another individual or, a company may have in its database, private information of someone. Individuals know that an infon is in possession of somebody else. Infons may belong to individuals. Finally, an infon is owned by an individual. The ownership is the basis for the specification of privacy.

The owner sovereignty principle restrains the right or sovereignty of people over their owned infons. A policy supporting the owner sovereignty principle restrains the possessor in the role of ‘content and topic observer’ and preserves the owner in the role of ‘informed owner’ and ‘refresher’. The contract between owner and possessor restricts the possibilities and rights of the possessor for using content and topics on an ongoing basis by additional actions such as

- to monitor activities of the possessor,
- to collect information (about conditions of possession),
- to give a warning to the owner, and
- to take actions such as use, security, welfare, accuracy, correctness, and maintenance of infons to the owner.

The collaboration is faithful if the portfolio and profile of contracting possessor do not include any forbidden action or ability, all reporting obligations are observed, and the proprietor is able to observe obligations applied to the possessor.

The private database is called information wallet if it is a component service with the following additional function enhancements for owners o, possessors p, infons i, infon requests ri, time stamps t, delivered infon streams identifiers si, public keys puk(ri, o, p, t) for p, private keys prik(i, o, p, t) for o, records of delivered infons by the owner store(o, i, p, si), and encoding and decoding relations encrypt(i, prik, si), decrypt(p, ri, si, puk, t) extended by steganographic watermarking mark(i, o, p) for infons:

- satisfy(request(ri, o, p, t)) ⇒ encrypt(i, prik(i, o, p, t), si)) ∧ deliver(p, o, si)) ∧ store(o, i, p, si))
- decrypt(p, ri, si, puk(si, o, p, t), t') ⇒ inform(o, Act(p, si, decrypt), t') ∧ mark(i, o, p)
- read(p, mark(i, o, p), t') ⇒ inform(o, Act(p, si, read), t')
- send(p, mark(i, o, p), p', t') ⇒ inform(o, Act(p, si, send(p, p')), t') ∧ ¬send(p, mark(i, o, p), p', t') ∧ send(p, ri, p', t')
We assume that watermarked infons cannot be changed by anybody. We can show now that information wallets preserve the owner sovereignty principle.
Development of Collaboration Frameworks for Distributed Web Information Systems

Klaus-Dieter Schewe1, and Bernhard Thalheim2

Abstract
Specification of distribution has neglected over a long period. Instead of explicit specification of distribution, multi-database systems and federated database systems have been extensively discussed in the literature. From the other side, database research has succeeded in developing approaches that incorporate conceptual specification and allow to reason on systems at a far higher abstraction level. With the advent of web information systems systems became naturally distributed. Therefore, we need a techniques for conceptual description of distribution. Distribution does not stand alone but follows computations and business needs. Thus, we need to consider structuring, functionality and distribution at the same time. Since these aspects are intertwined with each other and systems cooperate, communicate and coordinate their action we base our consideration on collaboration. It integrates communication, coordination and cooperation. In this paper we develop a specification framework for collaborating systems.

1 Introduction
1.1 Challenges Imposed By Collaborating Communities
The WWW has changed the way computational devices might be used. Currently, the main bottleneck of the web is not the communication bottleneck but the search bottleneck. Communities of leisure, communities of work, and communities of interest share their information space depending on their tasks instead of becoming lost while seeking information in the “World-Wide Wastebasket” Collaboration in general requires more sophisticated information structures that include meta-information at a variety of levels including service quality levels. This allows to locate information structured and stored by other parties, to trace the change of the information. In this case, parties use a “global yet personal information system”. Completeness of knowledge on the information space is not the main challenge if meta-information may be exchanged among collaborating parties.

Ubiquitous computing becomes a paradigm for classical computation devices and goes far beyond classical embedded systems computing. Users and systems require another collaboration principles than those developed in the past. Ubiquitous systems require sophisticated support in mobility for devices, services, users, and networks, require context awareness within a wide range of changing situations, and deep support for collaborations among groups of people and systems. The last support must be based on facilities for conferencing and communicating as well as on facilities for storage, maintenance, delivery, and presentation of shared data, shared functions, and shared control. Collaborations may be performed in real-time or asynchronous. Additionally, access and tracing of past activities is required.

Collaboration adds to modeling a new dimension: location. Location is not of importance for stationary devices. It is based on special data structures in which location information can be encoded and efficiently stored and in which the dynamic position of objects, their availability, their service level etc. can be maintained.

Collaboration is also based on context-awareness, i.e. on representation of user needs and demands, of roles of users, of portfolio of users or groups of users, and of user profiles. Collaboration is based on dynamic and partially ad-hoc grouping of people and systems. In this case, collaboration also requires calibration and adaptation of systems to changing situations. Finally, collaboration must be based on synchronization and on consistency support since it is based on shared data that might be created, modified, and deleted. Consistency support may be based on contracts contracted by collaborating parties. These contract may, for instance, require certain mechanisms for data recharging and data synchronization depending on profiles and portfolio.

1.2 New Paradigms Raised by Collaborating Communities
Collaboration requires a change in computing paradigms beyond programming that can be based on Hoare rule semantics [Alonso et al., 2004]. Classical imperative programming uses tough and restrictive facilities of control. The way of computation may vary depending on the collaborating party. Collaboration is based on interference or more general on concurrency. Therefore, compositional development of languages cannot be maintained longer. We may use the SiteLang [Düsterhöft and Thalheim, 2001] storyboarding language instead. It provides different conditions for steps
such as accept-on conditions [Srinivasa, 2000] or more generally rely-conditions and guarantee-conditions to both pre- and post-conditions. Rely conditions state what can be tolerated by the party. Guarantee-conditions record the interference that other processes will have to put up with if they are allowed to run in parallel. We envision in this paper that these conditions can be generalized to a specific style of assumption-commitment specification.

Collaboration has often been restricted to communication or communication-based concurrency. The distinction between this kind of concurrency and state-based concurrency cannot be used since collaboration also includes cooperation that requires

Collaborating communities are often self-organizing. The organization is context-dependent and emergent. So, the organization of the community itself must be reflected by the collaboration itself.

Collaboration uses a more elaborated concept of locality. Each party may use a specific communication interface that is partially agreed with other parties, may apply a number of restrictions to each of the parties, and may insist of a number of obligations that must be fulfilled by the parties. The local systems may be modeled as collaborating components [Thalheim, 2002]. We also consider small and evolving components.

With the advent of web information systems massively collaborating systems are developed and compete with classical systems. They are succeeding whenever ‘swarm intelligence’ outperforms better, whenever partnership models based on senior and junior parties are clearly specified, and whenever collaboration appears on demand, e.g., on contract, on force, on availability, or on interest, e.g., on desire, on interest or pleasure for groups of leisure.

A number of architectures have already been proposed in the past for massively distributed and collaborating systems. In the sequel we use the 3K model for specification of distribution and collaboration. Collaboration is going to supported on the basis of exchange frames and information service [Lockemann, 2003]. The first specify dissemination, e.g. announcement, volume, time, and protocols. The latter are use for specification of the information service with extraction, transformation, load, and representation. Such distributed services are based on classical communication facilities such as routing, e.g., P2P like with query based network propagation), such as subnetting and propagation.

2 Explicit Specification of Collaboration

According to [Safra et al., 2003], collaboration means to work jointly with others or together especially in an intellectual endeavor and to cooperate with an agency or instrumentality with which one is not immediately connected. Communication is used in a variety of facets as an act or instance of transmitting or a process by which information is exchanged between individuals through a common system of symbols, signs, or behavior. Coordination expresses the act or action of coordinating the harmonious functioning of parts for effective results. Cooperation expresses the action of cooperating. This understanding has directly led to the 3K model of collaboration that is the basis of our understanding of collaboration.

2.1 Conceptual Modeling of Collaboration

We may now use this model for specification of the three perspectives of collaboration displayed in Figure 1:

**Communication** is defined via exchange of messages and information or classically defined via services and protocols [König, 2003]. It depends on the choice of media, transmission modes, meta-information, conversation structure and paths, and on the restriction policy.

**Coordination** is specified via management of individuals, their activities and resources. It is the the dominating perspective of collaboration. The specification is based on the pre-/post-articulation of tasks and on the description management of tasks, objects, and time. Coordination may be based on loosely or tightly integrated activities, may be enabled, forced, or blocked.

**Cooperation** is the production taking place on a shared space. It can be considered as the workflow or life case perspective. We may use a specification based on storyboard-based interaction [Srinivasa, 2000] that mapped to (generic and structured) workflows. The information exchange is based on media types for production, manipulation, organization of contributions.

We use these ingredients of the perspectives for the specification of collaboration.

![Figure 1: The collaboration triangle relating communication, coordination, and cooperation](image)

A number of models have already been proposed for CSCW systems such as coordination theory [Malone and Crowston, 1994], activity theory [Kaptelinin et al., 1995], task management approaches [Kreifelts et al., 1999], action/interaction theory [Fitzpatrick et al., 1995], and object-oriented conceptual models [Tege, 1996]. We generalize these approaches and propose a more general model. We find dependent views on the diagram in Figure 1:

**Communication act view** which is based on sending and receiving collaboration acts;

**Concurrency view** which is based on commonly used data, functions, and tools;

**Cooperation context view** that combines the context of cooperation, i.e. portfolio to be fulfilled, the cooperation story and the resources that are used.
2.2 The Collaboration Style and Pattern

The collaboration style is based on four components describing supporting programs, the information system including session management, user management, and payment or billing systems; data access pattern for data release through the net, e.g., broadcast or P2P, for sharing of resources either based on transaction, consensus, and recovery models or based on replication with fault management, and for remote access including scheduling of access; the style of collaboration, on the basis of peer-to-peer models or component models or push-event models which restrict possible communication; and the coordination workflows, describing the interplay among parties, discourse types, name space mappings, and rules for collaboration.

Collaboration pattern generalize protocols and their specification [König, 2003]. They include the description of parties, their responsibilities, roles and rights. We know a number of collaboration pattern supporting access and configuration (wrapper facade, component configuration, interception, extension interface), event processing (reactor, proctor, asynchronous completion token, accept connector), synchronization (scoped locking, strategized locking, thread-safe interface, double-checked locking optimization) and parallel execution (active object, monitor object, half-sync/half-async, leader/followers, thread-specific storage): Proxy collaboration uses partial system copies (remote proxy, protection proxy, cache proxy, synchronization proxy, etc.). Broker collaboration supports coordination of communication either directly, through message passing, based on trading paradigms, by adapter-broker systems, or callback-broker systems. Master/slave collaboration uses tight replication in various application scenarios (fault tolerance, parallel execution, precision improvement; as processes, threads; with(out) coordination). Client/dispatcher collaboration is based on name spaces and mappings. Publisher/subscriber collaboration is also known as the observer-dependent paradigm. It may use active subscribers or passive ones. Subscribes have their subscription profile. Model/view/controller collaboration is similar to the three-layer architecture of database systems. Views and controllers define the interfaces.

2.3 Portfolio and Task Specification

A portfolio is determined by the responsibilities one has and is based on a number of targets one has. The party portfolio within an application is thus based on a set of tasks a party has or intents to complete and for which solution the party has the authority and control over, a description of involvement within the task solutions, and a collaboration that is formed for the tasks solution.

Task modelling means to understand what a user want to accomplish while visiting the web information system. At the same time, task analysis may lead to a reorganization of the work processes to be supported. Task analysis leads to a description of things users do, of things they act on, and of things they need to know. It does not specify how the task is accomplished. The tasks need to be representative for the application, important within the application, and completely supported. Task support can be tailored depending on the profile and the context of the parties.

Collaborations are formed according to tasks to be solved. Each of the parties has a portfolio that consists of all tasks and that defines the involvement, collaboration and restrictions. The specification of party (and user) portfolio is based on the following specification frame:

- **Party portfolio**: (party portfolio name)
  - **Task**: (general description)
    - **Characterization**: (general description)
      - **Initial state**: (characterization of initial states)
      - **Target state**: (characterization of target states)
    - **Profile**: (profile presuposed for solution)
      - **Instruments**: (list of instruments for solution)
        - **Auxillary**: (list of auxiliary conditions)
        - **Collaboration**: (collaboration style/pattern)
          - **Execution**: (list of activities, control, data)
        - **Result**: (final state, target conditions)
    - **Party involvement**: (general description)
      - **Role**: (description of role)
    - **Part**: (behavioral categories/stereotypes)
      - **Collaboration**: (general description)
        - **Communication**: (protocols, services and exchange)
          - **Coordination**: (contracts and enforcement)
          - **Cooperation**: (flow of work)
        - **Restrictions**: (general description)
          - **Party restrictions**: (general description)
          - **Environment**: (general description)

2.4 Cooperation Specification through SiteLang

We distinguish between the execution of the computer system and the execution of the interaction engine [Goldin et al., 2000]. The first execution is specified through workflows and describes the stepwise execution at the computational device. The second execution describes how the user recognizes the system behavior. Since we are mainly interested in the interaction model we concentrate in this paper on the specification of party or user interaction. We thus use a model that has already been proposed and widely applied for description of web information systems. The language allows to express stories. The story of interaction with the information system is the intrigue or plot of a narrative work or an account of events. The story space consists of a well-integrated set of stories and can be modeled by many-dimensional (multi-layered) graphs.

A story is a run through the story space by a collaborating set of parties. A story is composed of scenes. Each scene belongs to a general activity. Basic dialogue scenes may be combined to complex dialogue scene based on algebraic operations \( \square \) (choice), \( \| \) (parallel execution), \( ; \) (sequential ex-
execution), and \((.)^+\) (iteration). We may derive extended operations such as simple iteration \((.\rangle^+\) and optional execution \(\square\text{skip}\). Complex dialogue scenes are represented by frame boxes. We represent basic dialogue scenes by ellipses. The transitions among dialogue scenes are represented by curves.

**Example 1** Exercises considered so far in e-learning environments are often single-choice or multiple-choice exercises. These exercises and examinations constitute only a very small portion of possible exercises and examination tasks.

Using the story boarding language we can represent the scene supporting collaborative solution of exercises by the following expression:

\[
\langle T; ( ( D \square (C \land P)) \parallel ( ( I \land U )); H; (R; H)^+); A; ((S \square \text{skip}); E; I; N; H; (R; H)^+); A)^+; S \rangle
\]

with the dialogue stages \(T\) (Task delivery stage), \(D\) (Delivery of prepared data), \(C\) (Collection of users data), \(I\) (Information on applicable algorithms), \(P\) (Preparation of learners data), \(U\) (Code upload and installation), \(H\) (Formulation of learners hypotheses), \(A\) (Computation of association through mining), \(S\) (Submission of competitive solution), \(E\) (Evaluation of submitted solution), \(I\) (Inspection of sample solution and comparison with evaluations of competitors), \(N\) (Preparation for next trial for solution), and \(R\) (Reminder on learning element on hypothesis). Further, the symbols \(\langle\rangle\) and \(\rangle\) are used for denoting the entry stage and the termination stage of the scene.

Cooperation specifies

- the services provided, i.e., informational processes consisting of views of the source databases, the services manager supporting functionality and quality of services, and the competence of a service manifested in the set of tasks that may be performed, and
- requirements for quality of service.

### 2.5 Coordination Specification and Contracts

Coordination supports the consistency of work products, of work progress, and is supported by an explicitly specified coordinator. If work history is of interest, a version manager is integrated into the exchange support system. The coordination is supported by an infrastructure component. The coordination component observes modification of data that are of common interest to collaborating parties and resolves potential conflicts. The conflict resolution strategy is based on a cooperation contract. The contract is global to all parties and may contain extensions for peer-to-peer collaboration of some of the parties.

Coordination is based on a coordination contract. The contract consists of

- the coordination party characterization, their roles, rights and relations,
- the organization frames of coordination specifying the time and schema, the synchronization frame, the coordination workflow frame, and the task distribution frame,
- the context of coordination, and
- the quality requirements (ubiquity, security, interpretability, consistency, view consistency, scalability, durability, robustness, performance) for coordination.

**Contract:** (name)
- **Based on:** general conditions
- **Parties:** (general description)
- **Proprietor:** (...)
- **Possessor:** (...)
- **Trustee:** (...)
- **Arbiter:** (...)
- **Subject matter:** (Media object suite)
- **Exchange:** (binding obligations, permissions)
- **Computation:** (obligations, permissions)
- **Distribution:** (obligations, permissions)
- **Monitoring:** (managers: recognizer,)
- **Notification:** (states, timer, constraint scanner)
- **Correlation:** (protocols, obligations, permissions)
- **Considerations:** (legal conditions)
- **Enforcement:** (actions, termination)

We distinguish four levels of coordination specification. The syntactical level uses an IDL description and may use coordination constructs of programming languages. We use constructs of the JDL (job description language) for this description of resources, obligations, permissions, and restrictions. The behavior level specifies failure-atomicity-, execution-atomicity- pre-, rely-, guarantee- and post-conditions. The synchronization level specifies service object synchronization and paths and maintains a synchronization counter or monitor. The fourth level specifies quality of services level.

The coordination profile is specified by a coordination contract, a coordination workspace, synchronization profile, coordination workflow, and task distribution.

**Coordination profile:** (name)
- Based on: general conditions
- **Formation:** (general description)
- **Contract:** ...
- **Lifespan:** ...
- **Contract variant:** ...
- **Parties:** (names)
- **Organization:** (names, general description)
- **Infrastructure:** (name, general description)

The infrastructure of parties is characterized as follows:

**Infrastructure:** (name)
- **Workspace:** ...
- **Support:** ...

We distinguish between the frame for coordination and the actual coordination. Any actual coordination is an instance of the frame. It uses additionally an infrastructure. The contract specifies the general properties of coordination. Several variants of coordination may be proposed. The formation of a coordination may be based on a specific infrastructure. For instance, the washer may provide a workspace and additional functionality to the collaborating parties.

### 2.6 Party Specification

The party specification is based on the party profile, the organizations, the parties portfolio given above, and the infrastructure characterization:
3 Distribution Frameworks Supporting Collaboration

Specification of distributed information systems has neglected over a long period. Instead of explicit specification of distribution different collaborating approaches have been tried such as multi-database systems, federated database systems. Classically, distribution is tackled on the basis of services. Services are usually investigating on one of the (seven) layers of communication systems. They are characterized by two parameters: Functionality and quality of service. Structuring has been in the past out of the scope.

Distributivity is defined in this paper by the pair (Services, Exchange Frames).

Communication contracts specify the collaboration architecture and the style of exchange.

3.1 Services

A service consists of a media type, the characteristics provided and properties guaranteeing service quality and is defined by the quadruple (Media type, Service Manager, Competence, Characteristics), i.e., \( S = (M, Man, C, F) \).

Media types offer their own functions including statistical packages, functions proposed for data warehouses, or data mining algorithms. The services manager \( Man \) supports functionality and quality of services and manages containers, their play-out and their delivery to the client. It is referred to as a service provider. The competence of a service manifests itself in the set of tasks \( T \) that may be performed and the guarantees for their quality.
The different aspects of collaborating systems may be represented similar to Figure 2 and managed by data structures displayed in Figure 3. The external components, such as the work sessions and the session manager, belong to the coordination layer. They show how one coordination component can be linked to the components of the communication layer. The communication infrastructure interacts with the user interface and background processes through the event handler. The user buffer provides temporary storage of messages and is used for synchronization of data exchange.

4 Conclusion

The research reported in this paper aims in the development of a specification framework. Currently, no tool set is available for the specification of collaboration. UML diagramming facilities may be used for the specification. We however prefer more rigid and better based specification methods and thus turned to database specification techniques backed by ASM theory [Börger and Stärk, 2003].

The approach to specification has already been applied in one e-government platform that supports collaboration among parliamentarians, collaboration within and among political parties and groups, collaboration for development of (juridical) documents, and collaboration among parliamen-

tarians and citizens. Therefore, we consider the framework as a good option for collaboration platforms.

References


Figure 4: A generic model for a collaboration management system

(Appendix: Only for illustration)

Figure 5: The integration maintenance system of a collaborating system
Visual SQL:
Towards ER-Based Object-Relational Database
Querying

Bernhard Thalheim
Computer Science Institute, Christian-Albrechts-University Kiel,
Olshausenstrasse 40, 24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de

Database querying and programming based on Visual SQL

Query formulation is still a difficult task whenever a database schema is large or complex. The user has to entirely understand the schema before a correct and complete formulation of the query will be found. Furthermore, users may overlook types in the SQL schema that must be used in the query. Visualization based on Visual SQL leads to higher conceptual correctness and conceptual completeness.

Visual SQL is at the same time

• as powerful as SQL-2 and SQL:1999,
• is well-founded and has a well-defined semantics [Tha03],
• simpler to use and to comprehend, and
• less error-prone in complex settings.

Visual SQL shows what we would gain after realization of Chen’s dream on ER database systems. It demonstrates the power of visual programming already for existing database technology. There have already been reported several project (for the analysis of these projects, proposals and tools see [Tha03]) but none of them covered the complete SQL:1999 or SQL-2 standard. It is more powerful than the editors for MS Access and Oracle.

Visual SQL eases usability (understandability, learnability, operability, attractiveness) [JT03]. Users do not have to have the ability to formulate a query while having complete understanding of a large database schema, of the impact of specific values such as null values and of the integrity constraints.

The Cottbus and Kiel teams have developed the Visual SQL editor and a retranslator from SQL-2 to Visual SQL. The editor has been used in a number of
projects and for teaching purposes in universities and high schools in Germany and New Zealand. It has a German, English and Chinese version. The database schema is typically given on the basis of DBMain schemata. The re-translator has been used in project aiming to document already existing SQL code. The largest SQL query that has been cracked through the re-translator consists of more than 250 dense lines of SQL code. The version 1.5 of the system has been exhibited at CeBIT 2006. The version 1.6 is available from teaching path of the website http://www.is.informatik.uni-kiel.de/~fiedler/.

A demonstration example

Let us compare the facilities of Visual SQL with SQL-92 based on the database schema used in [Tha00].

We consider the following query:

Provide data on students who have successfully completed those courses which have successfully been given or which are currently given by the student’s supervisor?

Figure 1: Comparison of Visual SQL query formulation and SQL-2 representation

References


Abstract

Traditional software engineering and information systems engineering is structured into requirements analysis and definition, systems design, systems implementation and testing, and systems operation and maintenance. For web information systems the traditional approach suffers in three obstacles: late integration of architectural decisions, neglecting user expectations, and late implementations. Web information systems follow pre-defined three-tiered architectures. They also must consider expectations, profiles and portfolio of a large variety of users. Additionally, users expect an early involvement into the development and an early evaluation of the system. At the same time, traditional software engineering has led to quality assurance during the development process based on CMMI and SPICE. This paper shows how the achievements of traditional software engineering can be preserved while overcoming the obstacles. We develop an approach that integrates application domain description with development of presentation and information systems. The development methodology is improved based on the requirements of SPICE.

1. Introduction

1.1 Web Information Systems

In general, every data-intensive information system that is realised in a way that users can access it via web browsers or thin clients will be called a web information system (WIS). Typically, such systems are realised by a web site, i.e. a collection of web pages, each of which is just a file that can be interpreted by the browser. We may thus apply classical methods of information systems engineering to website development. The main difference is however that the purpose and usage of the systems as well as the users are known in advance. Therefore, we have to pay more attention to a “mission statement” for the system, i.e. the important questions “Who will use the system?”, “Which user intentions and behaviour shall be supported?”, “Which technical devices will be used by the users?”, etc. have to be taken into account. We thus consider six different aspects for web information systems: intention, usage, content, functionality, context, and presentation.

1.2 Engineering of Web Information Systems

Bjørner [2] and Heinrich [6] consider software engineering on the basis of the trilogy consisting of the application domain description, the requirements prescriptions, and finally the systems specifications. They extend modern software engineering approaches, e.g. [11, 15] by explicit consideration of the application domain. Schewe and Thalheim [16, 19] extend these approaches by (1) explicit consideration of user expectations, profiles and portfolio and by (2) storyboards and story spaces. WIS specification is oriented towards systems that are easy and intuitive to use. Therefore the classical information systems development model adds to systems development the development of presentation systems, too. Classically user interfaces are built after the system has been developed. In this case, the user has to learn how the system behaves and must adapt his/her behaviour to the systems behaviour. We repair this mismatch by primarily considering the user worlds, the user stories, and the applications.

WIS has two different faces: the systems perspective and the user perspective. These perspectives are tightly related to each other. We consider the presentation system as an integral part of WIS. It satisfies all user requirements, and it is based on real life cases.

Top-down development of systems seems to be the most appropriate whenever a system is developed from scratch or a system is extended. For this reason, we may differentiate among three layers: the systems description
and prescription layer, the conceptual specification layer, and the systems layer. These layers may be extended by the strategic layer that describes the general intention of the system, by the business user layer that describes how business users will see the system and by the logical layer that relates the conceptual layer to the systems layer by using the systems languages for programming and specification.

Classical software engineering typically climbs down to the implementation layer in order to create a productive system. The usual way in today’s WIS development is a manual approach: human modelling experts interpret the specification to enrich and transform it. This way of developing specifications is error-prone: even if the specification on a certain layer is given in a formal language, the modelling expert as a human being will not interpret it in a formal way. Misinterpretations, misunderstandings, and therefore the loss of already specified system properties is the usual business.

To overcome these obstacles a methodology for WIS engineering (WIS-E) has been developed [19]. The methodology is under continuous development. To prioritize the efforts and to ensure coherence of the enhancements, a comparison with more mature software engineering approaches was considered to be of benefit.

1.3 Software process improvement

Approaches to improve a software process can be based on e.g. modelling, assessment, measurement, technology adoption or management. The approaches supplement each other, but one of them is usually in a dominating position. Process assessment is a norm-based approach, which usually leads to evolutionary process improvement [1].

The development of web information systems is a special case for a software process. Such a process is considered to be the set of activities, methods, and practices used in the production and evolution of software [7] and the associated products [12]. A process transforms its inputs to outputs. The properties of the process have a great influence on its efficiency and on the properties of its outputs. It is widely accepted that the quality of a software product is largely determined by the quality of the process used [21]. The quality of a process can be measured using the attributes of process capability.

Since 1998 the technical report ISO/IEC TR 15504 [9] has provided a framework for the assessment of software processes. It is generally known as SPICE (Software Process Improvement and Capability dEtermination), a term, which originally stands for the initiative to support the development of a standard for software process assessment. The international standard ISO/IEC 15504, with five parts, has been published in 2003-2006 [9, 10].

Process capability is a characterization of the ability of a process to meet current or projected business goals [8]. Part 2 of the standard defines a measurement framework for the assessment of process capability. In the framework, Process capability is defined on a six point ordinal scale (0-5) that enables capability to be assessed from Incomplete (0) to Optimizing (5). The scale represents increasing capability of the implemented process. The measure of capability is based upon a set of process attributes, which define particular aspects of process capability [9].

1.4 Scope of the study

This paper introduces the background for WIS engineering and its improvement using SPICE. WIS must provide a sophisticated support for a large variety of users, a large variety of usage stories, and for different (technical) environments. Due to this flexibility the development of WIS differs from the development of information systems by careful elaboration of the application domain, by adaptation to users, stories, environments etc. In this context, process assessments and interpretation of the software engineering process models are used to improve a WIS engineering methodology.

The next section explains the process of methodology improvement and the general approach. Section 3 describes one of the major novel steps for WIS engineering based on the SPICE analysis: The application domain description. Section 4 concludes our experiences in supporting WIS engineering with SPICE assessments.

2. Towards managed Web Information Systems engineering

2.1 Process capability in process improvement context

The starting point for software process improvement actions is the gap between the current state of an organization and the desired future state [5]. These two states can be characterized using a norm for good software practices. The most popular among such norms are the Software Engineering Institute’s CMMI [3] and SPICE [10] by international standardization bodies ISO/IEC. The current state of software practice can be evaluated with the help of process assessments, which is a disciplined appraisal of an organization’s software process by utilizing a norm: the software process assessment model.

2.2 Applying process improvement principles in method engineering
A methodology for WIS engineering (WIS-E) was evaluated using a standard assessment approach with the aim to generate improvement ideas for the development of the methodology. The purpose of using an assessment approach was to improve the structure of WIS-E and its practice definitions. The assessment was performed (1) to check that corresponding elements of 15504-5 process assessment model can be found in WIS-E; (2) to ensure that future use of WIS-E would satisfy the requirements of the assessment model and; (3) to check the assessment model scope from a database development oriented point of view.

The assessment was performed in several phases during a long period of time. The assessment team consisted of the Co-Design chief engineer and two experienced assessors. The main steps of the assessment were planning, examination, discussion, documentation and review. During the planning, the scope of the assessment was fixed and the necessary material selected. The material that described the WIS-E was examined by the assessors and detailed questions were prepared. In the discussions within the assessment team the methodology was studied and the assessment questions were answered. The documentation of the assessment results provided a set of improvement ideas and a rough indication of the methodology’s correspondence to the capability requirements of the assessment model. During the review the assessment results were explained and the findings were discussed to create a consistent path for improvement and to set priorities.

2.3 Experiences of using SPICE in developing a methodology

In general, the assessment approach worked well and the findings were considered relevant. A difficulty in the assessment was that the methodology was developed at the same time. The extended assessment period and the pressure to modify the methodology based on the preliminary discussions, made some findings obsolete.

In the beginning of the assessment the main issue was to map the Co-Design processes to the processes of the 15504-5 assessment model. E.g. the Stakeholder contract step contained elements of three model processes i.e. Requirements elicitation, Contract agreement and Project management. Besides the structural issues, also the content of the WIS-E practices and work products was examined and discussed.

During the reviews concepts were clarified and redundant descriptive elements removed, also the wording of the elements was harmonized. The main benefit for WIS-E was the improvement of the structure. The result can be seen in clarified statements of process purposes and outcomes, and defined work products. A structured methodology is easy to develop further, the completeness and correctness of the methodology can be ensured, and the overall quality of the method is improved.

When considering the content of the methodology the software engineering process assessment approach provided only minor improvements. On the contrary, the assessment model could be further developed based on the Co-Design approach. The most prominent areas are the database design and the consideration of the user needs in wider scope and from various aspects, than what the prevailing software engineering process models cover.

2.4 Implications for WIS engineering

We applied the SPICE framework for improving WIS engineering, for systematic development of all facets, aspects, and work products, and for orchestration of WIS specifications. Web information systems engineering models the activities performed or managed by people participating in the engineering process. Models support understanding of the application domain and of the WIS, can be a source of inspirations, can be used for presentation and training, are abstractions that assert and predict behaviour of the WIS, and are the source for the implementation of the web information system.

![Image](image_url)

**Fig. 1. The primary dimensions of information systems engineering**

Activities may result in work products, may revise existing work products, or may be based on different work products. Work products considered in our framework are the developed documents and their role to the completion of the activity (postcondition).

We therefore distinguish between the five primary dimensions of WIS engineering. Activities (“how”) describe the way how the work is performed and the practices for the work. Work products (“what”) are the
result of the specification and are used during specification. Roles ("who") describe obligations and permissions, or the involvement of actors in the specification process. Aspects ("where") are used for separation of concern during the specification process. Resources ("on which basis") are the basis for the specification. These five main dimensions are hierarchically structured. We structure the associations among the dimensions in Figure 1.

These five dimensions must be mapped to technology. This mapping includes the derivation of the general solution and of the software architecture. The five dimensions can be enhanced by secondary dimensions. Finally, each object of one dimension may be associated to other objects of the same or other dimensions. We observe in the resource dimension, that, for instance, methods are founded on theories.

The Co-design approach to modelling [16, 19] integrates specification of structuring (structure and static integrity constraints), functionality (processes and dynamic integrity constraints), distribution (services and exchange frames), and interactivity (foreseen stories and envisioned actors).

2.5 Orchestration of WIS development for managed engineering

Orchestration uses the metaphor to music. It denotes the arrangement of a musical composition for performance by an orchestra. It may also denote harmonious organization, i.e. through orchestration of cultural diversities. Figure 1 displays dimensions of information systems engineering. Partners must integrate all activities, for all aspects, for all resources, and all work products. SPICE [9] requires for development processes that the implemented process achieves its process purpose (SPICE level 1). SPICE level 2 is achieved if the process is implemented in a managed fashion (planned, monitored and adjusted) and its work products are appropriately established, controlled and maintained.

Therefore, managed engineering is based on performance management and on work product management. Performance management requires that objectives for the performance of the process are identified, performance of the process is planned and monitored, performance of the process is adjusted to meet plans, responsibilities and authorities for performing the process are defined, assigned and communicated, resources and information necessary for performing the process are identified, made available, allocated and used, and interfaces between the involved parties are managed to ensure both effective communication and also clear assignment of responsibility. Work product management requires a well-defined and well-implemented work product management. Requirements for the work products of the process must be defined as well as requirements for documentation and control of the work. Work products must be appropriately identified, documented, and controlled. Work products are going to be reviewed in accordance with planned arrangements and adjusted as necessary to meet requirements. [10]

We show now in the sequel how orchestration of WIS development leads to managed WIS engineering. Due to space limitations we restrict on the first process: Application domain description and requirements statement.

3. Application domain description for WIS

3.1 The Co-Design framework for WIS engineering

The co-design framework [18, 20] provides a methodology for description, prescription and specification. A methodology is the study of and knowledge about methods. Methods used in the co-design framework are based

(a) on the extended entity-relationship model that supports specification of structures and functions,

(b) on the storyboarding language SiteLang that supports specification of users, their profiles, their portfolio, their actions, and their stories and

(c) on the framework DistLang that provides a general framework for architecturing, for development of services and exchange frames.

We distinguish a number of facets or views on the application domain. Typical facets to be considered are business procedure and rule facets, intrinsic facets, support technology facets, management and organisation facets, script facets, and human behaviour.

In the sequel we demonstrate how SPICE-improved WIS engineering can be performed for the WIS application domain engineering. Application domain engineering forms a process in the sense of SPICE. It aims in describing the application perspective, i.e. the subject world, the usage world, the intentions of the WIS according to Pohl and Rolland [13, 14]. It results in a general statement of requirements. Requirements engineering aims in elicitation of requirements within the system environment, exploration of system choices, complete extraction of functional and non-functional requirements, conflict detection and resolution, documentation and negotiation of agreed requirements, and in providing a framework for WIS evolution.

Classical application domain engineering is rather informal and fuzzy. It is based on general multi-facetted descriptions of desires, beliefs, intentions, skills, and
The co-design framework is the first one that results in a formal description of the application domain. At the same time, the framework is based on a methodology. We describe the first process, Application domain description and requirements statement, of this methodology in the next subsection.

### 3.2 Application domain description and requirements statement

The most important outcome of application domain engineering is an application domain model and its associated application domain theory. The main activity is to gather from application domain business users, from literature and from our observations the knowledge about the application domain. It is combined with validation, i.e. the assurance that the application domain description commensurates with how the business users view the application domain. It also includes the application domain analysis, i.e. the study of application domain (rough) statements, the discovery of potential inconsistencies, conflicts and incompleteness with the aim of forming concepts from these statements.

**Process purpose: Goals and subject**

Each process of WIS development must be characterised by goals and the subject.

<table>
<thead>
<tr>
<th>Goals and subject</th>
<th>Application domain description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project scope: Milestones, financial issues</td>
<td>Agreement for development</td>
</tr>
<tr>
<td>Clarification of development goals (intentions, rationale)</td>
<td>Sketch of requirements</td>
</tr>
</tbody>
</table>

**Process Outcomes: Work Products as Process Results**

The work product is a result or deliverable of the execution of a process and includes services, systems (software and hardware) and processed materials. It has elements that satisfy one or more aspects of a process purpose and may be represented on various media (tangible and intangible).

Documents of the application domain layer are HERM [18] concept maps, HERM functionality feature descriptions, the DistrLang distribution specification resulting in contract sketches which include quality criteria, and the SiteLang interactivity specification of the application story with main application steps. The documents are combined within the Stakeholder contract and the feasibility study. Additionally a number of internal documents are developed such as life case studies, description of intentions, context specification, and user profiles and portfolio.

**Table 2. Developed documents**

<table>
<thead>
<tr>
<th>Application domain description section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information analysis</td>
</tr>
<tr>
<td>missions and goals of the WIS, brand of the WIS</td>
</tr>
<tr>
<td>general characterisation of tasks and users</td>
</tr>
<tr>
<td>general characterisation of content and functions</td>
</tr>
<tr>
<td>description of WIS context</td>
</tr>
<tr>
<td>Intensions of the web information system</td>
</tr>
<tr>
<td>catalogue of requirements</td>
</tr>
<tr>
<td>scenarios, scenes, actions, context and life cases</td>
</tr>
<tr>
<td>user profiles and user portfolio</td>
</tr>
<tr>
<td>actor profiles and actor portfolio, personas</td>
</tr>
<tr>
<td>Business rule model and storyboards</td>
</tr>
<tr>
<td>scenarios, scenes, and actions</td>
</tr>
<tr>
<td>life cases and context</td>
</tr>
<tr>
<td>WIS utilisation portfolio</td>
</tr>
<tr>
<td>scenarios, activities</td>
</tr>
<tr>
<td>supporting content and functions</td>
</tr>
<tr>
<td>non-functional requirements, context space</td>
</tr>
<tr>
<td>Metaphor description</td>
</tr>
<tr>
<td>base metaphors, overlay metaphors</td>
</tr>
<tr>
<td>metaphor integration and deployment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Official and contracting section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stakeholder contract</td>
</tr>
<tr>
<td>goal information</td>
</tr>
<tr>
<td>concept sketch, product functionality, story space, views on product data, view collaboration sketch</td>
</tr>
<tr>
<td>Comparison with products of competitors</td>
</tr>
<tr>
<td>Evaluation of development costs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Internal section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning of goals</td>
</tr>
<tr>
<td>development strategy and plan,</td>
</tr>
<tr>
<td>quality management</td>
</tr>
<tr>
<td>Development documents on product components with base practices, generic practices and capabilities, estimation of efforts</td>
</tr>
</tbody>
</table>

**Practises: Base Activities and Steps**

The system development process is usually decomposed into clearly distinguishable development activities that support to tackle separate issues at separate time. An activity either starts from scratch and results in a completed work product or starts from a set of work products and results in a completed or revised work product. Activities are work product transformations. Therefore, the steps can be given through refinement activities, i.e., elicitation, determination, instantiation, extension, and fitting. Activities typically proceed in one or (usually) more steps. Activities are describing the ways how the work is performed and can be practices...
according to SPICE. A practice contributes to achieving a specific development process purpose or to the achievement of a specific development process attribute or enhances the capability of a development process. Activities consist of steps. These activities are enhanced by acquisition activities and activities supporting the life cycle, which are not described here.

We envision three base activities and structure them into development steps:
1. Development of application domain description
2. Development of the stakeholder contract
3. Development of the internal documents such as description of product components and quality requirements with base activities, generic practices and capabilities and estimation of efforts etc.

We demonstrate the base activities for the first one in Table 3. We use the SiteLang specification language [17] and the conceptual framework of [16].

<table>
<thead>
<tr>
<th>Table 3. Steps of an activity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Development of application domain description</strong></td>
</tr>
<tr>
<td>1. Analyze strategic information</td>
</tr>
<tr>
<td>Specify mission and brand</td>
</tr>
<tr>
<td>Characterise in general tasks and users of application</td>
</tr>
<tr>
<td>Characterise in general content and functions domain</td>
</tr>
<tr>
<td>Describe WIS context</td>
</tr>
<tr>
<td>2. Derive intensions of the WIS,</td>
</tr>
<tr>
<td>Obtain general requirements</td>
</tr>
<tr>
<td>Extract life cases</td>
</tr>
<tr>
<td>Describe scenarios, scenes, actions, and context</td>
</tr>
<tr>
<td>Describe user profiles and user portfolio</td>
</tr>
<tr>
<td>Derive actor profiles and actor portfolio, personas</td>
</tr>
<tr>
<td>3. Extract business rule model and storyboards</td>
</tr>
<tr>
<td>Develop scenarios, scenes, and actions</td>
</tr>
<tr>
<td>Specify life cases and context</td>
</tr>
<tr>
<td>Eliciting metaphors</td>
</tr>
<tr>
<td>4. Revise business rules of the application</td>
</tr>
<tr>
<td>Possibly with reorganization models</td>
</tr>
<tr>
<td>5. Compare new business rules with known visions</td>
</tr>
<tr>
<td>6. Compare with products of competitors</td>
</tr>
<tr>
<td>7. Derive WIS utilisation</td>
</tr>
<tr>
<td>Describe scenarios to be supported</td>
</tr>
<tr>
<td>Describe activities based on word fields</td>
</tr>
<tr>
<td>Describe supporting content</td>
</tr>
<tr>
<td>Describe supporting functions</td>
</tr>
<tr>
<td>Describe non-functional requirements</td>
</tr>
<tr>
<td>Describe the context space</td>
</tr>
<tr>
<td>8. Develop metaphor</td>
</tr>
<tr>
<td>Describe base and overlay metaphors</td>
</tr>
<tr>
<td>Find metaphor integration</td>
</tr>
<tr>
<td>Develop templates for deployment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precondition</th>
<th>Contracted collaboration of all partners</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real necessity for WIS development</td>
</tr>
<tr>
<td>Postcondition</td>
<td>Description of application domain accepted and consistent</td>
</tr>
<tr>
<td></td>
<td>New business rule model accepted</td>
</tr>
</tbody>
</table>

Information analysis is based on WIS storyboard pragmatics. We usually start with templates that specify the brand of the WIS by the four dimensions provider, content, receiver, and main actions. The brand is based on the mission containing general statements on the content, the user, the tasks, purposes, and benefits. The content indication may be extended on the basis of the content item specification frame and by a rough description of context based on the general context template. The brand description is compared with the description of tasks to be supported. Tasks are specified through the general characterisation template. The specification of the intention of the WIS is based on the designation of goals of the WIS.

Additionally we integrate metaphors. They must be captured as early as possible. Metaphors may be based on the style guides that are provided by the customer. Metaphorical structures for WIS can be developed based on linguistic and cognitive research [4].

The application domain description feeds the derivation of the WIS utilisation portfolio. The general section of the WIS utilisation portfolio consists of a list of categories, refined brands, and a general description of kinds of WIS portfolio. The partner section of the WIS utilisation portfolio surveys partners, i.e. a list of actors with portfolio, possibly with rights, obligations, and roles. The storyboard section of the WIS utilisation portfolio describes scenarios used and activities which are based on a list of word fields characterising activities. Activities are characterised by an activity style, by activity pattern, and by collaboration styles and pattern. The content section of the WIS utilisation portfolio describes the content chunks together with the content portfolio, i.e. content demand, consumption, and production. The functionality section of the WIS utilisation portfolio describes supporting functions on the basis of function chunks and the functionality portfolio, i.e. demand, consumption, production of functionality. The WIS utilisation portfolio is enhanced by a designation of non-functional requirements, especially quality requirements and by a derivation of specifics of the context space.

4 Conclusions
Web information systems engineering inherits the achievements and approaches of information systems engineering. They differ in architectures that are used, in integration of suites of (information) systems, in demands for quality, in stories of utilisation etc. Therefore, WIS engineering must cope with novel requirements and lead to novel solutions.

The paper is one of the first contributions to explicit treatment of application domain descriptions for WIS. They become crucial for web information systems due to the general purpose use of websites by a large variety of users who use a large number of display clients who have very different profiles, goals, and tasks and who vary in their behaviour.

WIS engineering must be a process that is based on a well-defined methodology that is manageable. We applied a SPICE evaluation to our first methodology and improved this methodology based on the critics and findings. This paper discusses the improvement process and illustrates the results for the first engineering process.

Software process improvement applied to methodology development has potential, but is also demanding due to the fact that methodologies are more abstract than e.g. software projects. For instance, evidence in assessments can be interpreted quite freely. In this case, the main benefits were found in improving the structure of the methodology presentation.

WIS engineering is based on very early and stable integration of business users. These users do not want to control code or documents but are mainly interested in the result. Therefore, each specification step must lead to an executable specification.

References


Co-Design of Web Information Systems  
Supported by SPICE

Gunar Fiedler\textsuperscript{2}, Hannu Jaakkola\textsuperscript{1}, Timo Mäkinen\textsuperscript{1}, Bernhard Thalheim\textsuperscript{2}, Timo Varkoi\textsuperscript{1}\textsuperscript{1} Tampere University of Technology, P.O. Box 300, FI-28101 Pori, Finland\textsuperscript{2} Department of Computer Science, Kiel University, Olshausenstr. 40, 24098 Kiel, Germany
\{hannu.jaakkola, timo.makinen, timo.varkoi\}@tut.fi\textsuperscript{2} \{fiedler,thalheim\}@is.informatik.uni-kiel.de

Abstract. Web information systems (WIS) augment classical information systems by modern Web technologies. They require at the same time a careful development and support for the interaction or story spaces beside the classical support for the working space of users. These dimensions complicate the system development process. This paper shows how classical advanced methodologies can be carefully enhanced. We evaluated the Co-Design approach to information systems development according to the ISO/IEC 15504 Framework for Process Assessment (SPICE) and derived the potential and deficiencies of this approach. This evaluation has been used for a managed Co-Design methodology. Since WIS constantly change and evolve the development process never ends. We develop as a solution an optimization procedure and a runtime environment for the Co-Design approach that allows to cope with such changes, with evolution and extension of WIS and that demonstrates the new system facilities whenever a change is mastered.

1 Introduction

Planning, developing, distributing, and maintaining sophisticated large-scaled systems is one of the core competencies in software engineering. Properly working systems provide valuable assets to users as well as operators while erroneous, incomplete, and misused software causes losses in economical, technical, or social ways as systems become more and more ubiquitous.

Information systems are considered to be complex systems with a deep impact on the people’s daily life. Web information systems (WIS) are systems providing information to users by utilizing Web technologies. Usually, WIS are data-intensive applications which are backed by a database. While the development of information systems is seen as a complex process, Web information systems engineering (WIS-E) adds additional obstacles to this process because of technical and organizational specifics:

- WIS are open systems from any point of view. For example, the user dimension is a challenge. Although purpose and usage of the system can be formulated in advance, user characteristics cannot be completely predefined. Applications have to be intuitively usable because there cannot be training courses for the users. Non-functional properties of the application like ‘nice looking’ user interfaces are far more important compared with standard business software. WIS-E is not only restricted to enterprises but is also driven by an enthusiastic community fulfilling different goals with different tools.
• WIS are based on Web technologies and standards. Important aspects are only covered by RFCs because of the conception of the Internet. These (quasi-)standards usually reflect the ‘common sense’ only, while important aspects are handled individually.

• Looking at the complete infrastructure, a WIS contains software components with uncontrollable properties like faulty, incomplete, or individualistically implemented Web browsers.

• Base technologies and protocols for the Web were defined more than 10 years ago to fulfill the tasks of the World Wide Web as they had been considered at this time. For example, the HTTP protocol was defined to transfer hypertext documents to enable users to browse the Web. The nature of the Web changed significantly since these days, but there were only minor changes to protocols to keep Compatibility alive which is considered to be “indispensable”. Today, HTTP is used as a general purpose transfer protocol which is used as the backbone for complex interactive applications. Shortcomings like statelessness, loose coupling of client and server, or the restrictions of the request-response communication paradigm are covered by proprietary and heavy-weight frameworks on top of HTTP. Therefore, they are not covered by the standard and handled individually by the framework and the browser, e.g., session management. Small errors may cause unwanted or uncontrollable behavior of the whole application or even security risks.

WIS can be considered from two perspectives: the system perspective and the user perspective. These perspectives are tightly related to each other. We consider the presentation system as an integral part of WIS. It satisfies all user requirements. It is based on real life cases. Software engineering has divided properties into functional and non-functional properties, restrictions and pseudo-properties. This separation can be understood as a separation into essential properties and non-essential ones. If we consider the dichotomy of a WIS then this separation leads to a far more natural separation into information system requirements and presentation systems requirements. The system perspective considers properties such as performance, efficiency, maintainability, portability, and other classical functional and non-functional requirements. Typical presentation system requirements are usability, reliability, and requirements oriented to high quality in use, e.g., effectiveness, productivity, safety, privacy, and satisfaction. Safety and security are also considered to be restrictions since they specify undesired behavior of systems. Pseudo-properties are concerned with technological decisions such as language, middleware, operating system or are imposed by the user environment, the channel to be used, or the variety of client systems.

WIS must provide a sophisticated support for a large variety of users, a large variety of usage stories, and for different (technical) environments. Due to this flexibility the development of WIS differs from the development of information systems by careful elaboration of the application domain, by adaptation to users, stories, environments, etc.

Classical software engineering typically climbs down the system ladder to the implementation layer in order to create a productive system. The usual way in today’s WIS development is a manual approach: human modelling experts interpret the specification to enrich and transform it along the system ladder. This way of developing specifications is error-prone: even if the specification on a certain layer is given in a formal language, the modelling expert as a human being will not interpret it in a formal way. Misinterpretations, misunderstandings, and therefore the loss of already specified system properties is the usual business.

The paper is organized as follows: In section 2 we discuss existing approaches and methodologies for WIS development, especially the Co-Design approach with its integrated view on
structure, functionality, interactivity, and distribution. The presented methodologies miss central specifics in today’s WIS development. No methodology is of value if it is not supported and enforced by an organizational and technical infrastructure. The section ends with a short discussion of the international standard ISO/IEC 15504, which provides a framework for the assessment of software processes. Section 3 partially presents the results of an assessment of the WIS Co-Design approach to enable the implementation of ideas from ISO/IEC 15504 to develop the WIS Co-Design approach towards managed WIS engineering. Section 4 shows the application of principles of SPICE to WIS development within the Co-Design approach to prepare WIS-E processes to move towards the ‘Optimizing’ level.

2 Background and Related Work

Several methodologies and architectures were developed to cope with information systems engineering in general and WIS-E in particular. [KPRR03] provides an overview over concepts, methods, and tools in WIS-E as well as the relationships between classical software engineering and web development.

2.1 Classical (Web) Information Systems Methodologies

ARIS (Architecture of Integrated Information Systems, [Sch92]) defines a framework with five views (functional, organizational, data, product, controlling) and three layers (conceptual (‘Fachkonzept’), technical (‘DV-Konzept’), and implementation). ARIS was designed as a general architecture for information systems in enterprise environments. Therefore, it is too general to cover directly the specifics of Web information systems as they were mentioned in Section 1 and needs to be tailored.

The Rational Unified Process (RUP, [Kru98]) is an iterative methodology incorporating different interleaving development phases. RUP is backed by sets of development tools. RUP is strongly bound to the Unified Modelling Language (UML). Therefore, RUP limits the capabilities of customization. Like ARIS, RUP does not address the specifics of WIS-E. A similar discussion can be made for other general purpose approaches from software engineering [HSSM‘04].

OOHDM [SR98] is a methodology which deals with WIS-E specifics. It defines an iterative process with five subsequent activities: requirements gathering, conceptual design, navigational design, abstract interface design, and implementation. OOHDM considers Web Applications to be hypermedia applications. Therefore, it assumes an inherent navigational structure which is derived from the conceptual model of the application domain. This is a valid assumption for data-driven (hypermedia-driven) Web applications but does not fit the requirements for Web information systems with dominating interactive components (e.g., entertainment sites) or process-driven applications. There are several other methodologies similar to OOHDM. Like OOHDM, most of these methodologies agree in an iterative process with a strict top-down ordering of steps in each phase. Surprisingly, most of these methodologies consider the implementation step as an ‘obvious’ one which is done by the way, although specifics of Web applications cause several pitfalls for the unexperienced programmer especially in the implementation step. Knowledge management during the development cycles is usually neglected.

There are several methodologies that cope with personalization of WIS. For example, the HERA methodology [HBFV03] provides a model-driven specification framework for per-
sonalized WIS supporting automated generation of presentation for different channels, integration and transformation of distributed data and integration of Semantic Web technologies. Although some methodologies provide a solid ground for WIS-E, there is still a need for enhancing the possibilities for specifying the interaction space of the Web information system, especially interaction stories based on the portfolio of personal tasks and goals.

2.2 Co-Design of Web Information Systems

We distinguish a number of facets or views on the application domain. Typical facets to be considered are business procedure and rule facets, intrinsic facets, support technology facets, management and organization facets, script facets, and human behavior. These facets are combined into the following aspects that describe different separate concerns:

- The structural aspect deals with the data which is processed by the system. Schemata are developed which express the characteristics of data such as types, classes, or static integrity constraints.

- The functional aspect considers functions and processes of the application.

- The interactivity aspect describes the handling of the system by the user on the basis of foreseen stories for a number of envisioned actors and is based on media objects which are used to deliver the content of the database to users or to receive new content.

- The distribution aspect deals with the integration of different parts of the system which are (physically or logically) distributed by the explicit specification of services and exchange frames.

Each aspect provides different modelling languages which focus on specific needs. While higher layers are usually based on specifications in natural language, lower layers facilitate formally given modelling languages. For example, the classical WIS Co-Design approach uses the Higher-Order Entity Relationship Modelling language [Tha00] for modelling structures, transition systems and Abstract State Machines [BS03] for modelling functionality, Sitelang [TD01] for the specification of interactivity, and collaboration frames [Tha03] for expressing distribution. Other languages such as UML may be used depending on the skills of modelers and programmers involved in the development process.

A specification of a WIS consists of a specification for each aspect such that the combination of these specifications (the integrated specification) fulfills the given requirements. Integrated specifications are considered on different levels of abstraction (see Figure 1) while associations between specifications on different levels of abstraction reflect the progress of the development process as well as versions and variations of specifications.

Unfortunately, the given aspects are not orthogonal to each other in a mathematical sense. Different combinations of specifications for structure, functionality, interactivity, and distribution can be used to fulfill given requirements while the definition of the ‘best combination’ relies on non-functional parameters which are only partially given in a formal way. Especially the user perspective of a WIS contributes many informal and vague parameters possibly depending on intuition. For example, ordering an article in an online shop may be modelled as a workflow. Alternatively, the same situation may be modelled by storyboards for the dialog flow emphasizing the interactivity part. This principle of designing complex systems is called Co-Design, known from the design process of embedded systems where certain aspects can be realized alternatively in hardware or software (Hardware Software Co-Design).
Co-Design approach for WIS-E developed in [Tha00, Tha03, Tha04] defines the modelling spaces according to this perception.

We can identify two extremes of WIS development. *Turnkey development* is typically started from scratch in a response to a specific development call. *Commercial off-the-shelf development* is based on software and infrastructure whose functionality is decided upon by the makers of the software and the infrastructure than by the customers. A number of software engineering models has been proposed in the past: waterfall model, iterative models, rapid prototyping models, etc. The Co-Design approach can be integrated with all these methods.

At the same time, developers need certain flexibility during WIS engineering. Some information may not be available. We need to consider feedback loops for redoing work that has been considered to be complete. All dependencies and assumptions must be explicit in this case.

2.3 **Aligning Co-Design and the ISO/IEC 15504 Framework for Process Assessment**

There are several approaches for improving a software process, e.g. modelling, assessment, measurement, and technology adoption [SO98]. The approaches supplement each other, but one of them is usually in a dominating position. Process assessment is a norm-based approach, which usually leads to evolutionary process improvement [AAMN01]. The starting point for software improvement actions is the gap between the current state of an organization and the desired future state [GM97]. These two states can be characterized using a norm for good software practices. The most popular among such norms are the Software Engineering Institutes CMMI (Capability Maturity Model Integration) [CMM06] and SPICE (Software Process Improvement and Capability dEtermination) [ISO05], developed by international standardization bodies (ISO and IEC). The current state of software practice can be evaluated with the help of process assessments, which is a disciplined appraisal of an organizations software process by utilizing a norm: the software process assessment model. SPICE (the technical report ISO/IEC TR 15504) has provided a framework for the assessment of software processes since 1998. The term SPICE originally stands for the initiative to support
the development of a standard for software process assessment. The international standard
ISO/IEC 15504, with five parts, has been published in 2003-2006 to replace the set of technical
reports. Besides including a norm for a software process, SPICE can also be regarded as a
meta-norm that states requirements for software process assessment models. Figure 2 (from
[Bal98, p. 378]) depicts the context, where SPICE is applied.

Figure 2: Process Assessment, Improvement, and Capabilities

Process capability is a characterization of the ability of a process to meet current or pro-
jected business goals [ISO04]. Part 2 of the standard defines a measurement framework for
the assessment of process capability. In the framework, process capability is defined on a
six point ordinal scale (0-5) that enables capability to be assessed from Incomplete (0) to
Optimizing (5). The scale represents increasing capability of the implemented process. The
measure of capability is based upon a set of process attributes, which define particular as-
pects of process capability [ISO03]. For example, Level 1 (“Performed Process”) requires
that an implemented process achieves its process specific purpose. The level is characterized
by one process attribute (Process Performance), which presumes certain outcomes from the
process. Level 2 process is a Performed Process, which is also Managed i.e., implemented in
a managed fashion (planned, monitored and adjusted) and its work products are appropriately
established, controlled and maintained. Level 2 requirements are described by two process at-
methodology was examined using SPICE criteria with the aim to generate improvement ideas
for the methodology. A brief description of the initiative can be found in [FTJ+07]. The fol-
lowing section illustrates the implementation of basic Process Performance requirements in
Co-Design for WIS to form a basis for managed WIS engineering.

2.4 Requirements to Managed and Optimizable Co-Design for WIS

SPICE requires for managed development processes that the implemented process achieves
its process purpose (SPICE level 1). SPICE level 2 is achieved if the process is well-specified
and is implemented in a managed fashion (planned, monitored and adjusted) and its work
products are appropriately established, controlled and maintained. Therefore, managed en-
gineering is based on performance management and on work product management. Performance
management requires that

- objectives for the performance of the process are identified,
- performance of the process is planned and monitored,
- performance of the process is adjusted to meet plans,
- responsibilities and authorities for performing the process are defined, assigned and com-
  municated,
- resources and information necessary for performing the process are identified, made avail-
    able, allocated and used, and
• interfaces between the involved parties are managed to ensure both effective communication and also clear assignment of responsibility.

Work product management has to be well-defined and well-implemented. Requirements for the work products of the process must be defined as well as requirements for documentation and control of the work. Work products must be appropriately identified, documented, and controlled. Work products are going to be reviewed in accordance with planned arrangements and adjusted as necessary to meet requirements.

3 Orchestration of WIS Development for Managed Engineering

Developing WIS using the Co-Design approach can be seen as an orchestration of different specifications. Orchestration uses the metaphor to music. It denotes the arrangement of a musical composition for performance by an orchestra. It may also denote harmonious organization, i.e. through orchestration of cultural diversities.

We show now in the sequel how orchestration of WIS development leads to managed WIS engineering. Due to space limitations we restrict on the work products and activities of the first process: Application domain description and requirements statement. It aims in describing the application perspective, e.g., the subject world, the usage world, and the intentions of the WIS according to [Poh96, RSE04, Rol06]. It results in a general statement of requirements. Requirements engineering aims in elicitation of requirements within the system environment, exploration of system choices, complete extraction of functional and non-functional requirements, conflict detection and resolution, documentation and negotiation of agreed requirements, and in providing a framework for WIS evolution.

(WIS-E 1): Application Domain Description and Requirements Statement

The most important outcome of application domain engineering is an application domain model and its associated application domain theory. The main activity is to gather from application domain business users, from literature and from our observations the knowledge about the application domain. It is combined with validation, i.e. the assurance that the application domain description commensurates with how the business users view the application domain. It also includes the application domain analysis, e.g., the study of application domain (rough) statements, the discovery of potential inconsistencies, conflicts and incompleteness with the aim of forming concepts from these statements.

Process Purpose: Goals and Subject

<table>
<thead>
<tr>
<th>Goals and subject</th>
<th>Application domain description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Agreement for development</td>
</tr>
<tr>
<td></td>
<td>Project scope: Milestones, financial issues</td>
</tr>
<tr>
<td></td>
<td>Clarification of development goals (intentions, rationale)</td>
</tr>
<tr>
<td></td>
<td>Sketch of requirements</td>
</tr>
</tbody>
</table>

Process Outcomes: Work Products as Process Results

The work product is a result or deliverable of the execution of a process and includes services, systems (software and hardware) and processed materials. It has elements that satisfy one or more aspects of a process purpose and may be represented on various media (tangible and intangible).
Documents of the application domain layer are HERM [Tha00] concept maps, HERM functionality feature descriptions, the DistrLang distribution specification resulting in contract sketches which include quality criteria, and the Sitelang interactivity specification of the application story with main application steps. The documents are combined within the Stakeholder contract (‘Lastenheft’) and the feasibility study. Additionally, a number of internal documents are developed such as life case studies, description of intentions, context specification, and user profiles and portfolios.

<table>
<thead>
<tr>
<th>Developed documents</th>
<th>Stakeholder contract: goal information, concept sketch, product functionality, story space, views on product data, view collaboration sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Official and contracting section</td>
<td>Comparison with products of competitors, Evaluation of development costs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Developed documents</th>
<th>Information analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application domain description section</td>
<td>missions and goals of the WIS, brand of the WIS, general characterization of tasks and users, general characterization of content and functions, description of WIS context</td>
</tr>
<tr>
<td></td>
<td>Intensions of the web information system, catalog of requirements, scenarios, scenes, actions, context and life cases, user profiles and user portfolio, actor profiles and actor portfolio, personas</td>
</tr>
<tr>
<td></td>
<td>Business rule model and storyboards, scenarios, scenes, and actions, life cases and context</td>
</tr>
<tr>
<td></td>
<td>WIS utilization portfolio, scenarios, activities, supporting content and functions, non-functional requirements, context space</td>
</tr>
<tr>
<td></td>
<td>Metaphor description, base metaphors, overlay metaphors, metaphor integration and deployment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Developed documents</th>
<th>Planning of goals, development strategy and plan, quality management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal section</td>
<td>Development documents on product components and quality requirements with base practices, generic practices and capabilities, estimation of efforts</td>
</tr>
</tbody>
</table>

**Base Activities and Steps**

We envision three base activities: (1) Development of application domain description, (2) Development of the stakeholder contract, and (3) Development of the internal documents such as description of product components and quality requirements with base activities, generic practices and capabilities and estimation of efforts, etc.

We demonstrate the base activities for the first one. We use the Sitelang specification language [TD01] and the conceptual framework of [Tha05].
### Development of application domain description

1. Analyze strategic information
   - Specify mission and brand
   - Characterize in general tasks and users
   - Characterize in general content and functions
   - Describe WIS context
2. Derive intensions of the WIS,
   - Obtain general requirements
   - Extract life cases
   - Describe scenarios, scenes, actions, and context
   - Describe user profiles and user portfolio
   - Derive actor profiles and actor portfolio, personas
3. Extract business rule model and storyboards
   - Develop scenarios, scenes, and actions
   - Specify life cases and context
   - Eliciting metaphors
4. Revise business rules of the application
   - possibly with reorganization models
5. Compare new business rules with known visions
6. Compare with products of competitors
7. Derive WIS utilization
   - Describe scenarios to be supported
   - Describe activities based on word fields
   - Describe supporting content
   - Describe supporting functions
   - Describe non-functional requirements
   - Describe the context space
8. Develop metaphor
   - Describe base and overlay metaphors
   - Find metaphor integration
   - Develop templates for deployment

<table>
<thead>
<tr>
<th>Precondition</th>
<th>Contracted collaboration of all partners</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Real necessity</td>
</tr>
<tr>
<td>Postcondition</td>
<td>Descr. of application domain accepted and consistent</td>
</tr>
<tr>
<td></td>
<td>New business rule model accepted</td>
</tr>
</tbody>
</table>

Information analysis is based on WIS storyboard pragmatics. We usually start with templates that specify the brand of the WIS by the four dimensions provider, content, receiver, and main actions. The brand is based on the mission containing general statements on the content, the user, the tasks, purposes, and benefits. The content indication may be extended on the basis of the content item specification frame and by a rough description of context based on the general context template. The brand description is compared with the description of tasks to be supported. Tasks are specified through the general characterization template. The specification of the intention of the WIS is based on the designation of goals of the WIS. Additionally we integrate metaphors. They must be captured as early as possible. Metaphors may be based on the style guides that are provided by the customer.

### 4 SPICEing Co-Design: Towards Optimizing Processes

Every methodology is only valuable if it is supported and enforced by the technical and organizational environment. The methodology itself is only able to provide a frame for the quality management of each development process and the knowledge transfer between development
Imagine, you are requested to develop an application which enables registered users to search and browse through a photo library. Photos should be organized in categories while each photo might be bound to arbitrary categories. Categories are hierarchically ordered. Each photo is described by metadata like EXIF parameters. We choose an interactivity-driven design strategy. The client formulates the following specification: ‘The application should be accessible by all usual browsers. The user has to log in. After login he is offered to browse through the categories or search for photos based to the photo’s metadata. When the user selects a photo in a category or in the search result, the picture is shown in detail.’

Figure 3 shows a story space reflecting the given requirements. There are four scenes: the login scene for authenticating the user, the browsing scene for browsing the categories, the searching scene for searching photos and the viewing scene for looking at photos in detail.

Even this very small and abstract example reveals important shortcomings for top-down methodologies:

The story space is a valid one but does not reflect the intended application although it reflects the utterances of the client. The client describes runs through the story space, not the story space itself. The client will not explicitly mention the fact that a run may be cancelled or restarted because this is normal behavior. Because the client is usually not able to interpret abstract specifications, top-down approaches force the development team to go through the development steps down to implementation investing huge amounts of resources before the client will notice the error. Experienced modelers will notice this problem very soon. But nevertheless, the outcome of a managed development process has to be independent from personal skills.

It is not possible to implement the given storyboard as a Web application. There are additional scenarios in the story space because every browser offers the possibility to go back in the page history. These scenarios are not visible in the specification but cannot be excluded¹ (‘back button problem’). This error is only visible to the experienced modeler or during the tests after implementation.

There are artifacts on every layer of abstraction which are reused between different applications like the login scene in the given example. But there are also aspects of the application which are orthogonal to the abstraction layers. These aspects are no isolated modelling artifacts but determine the translation of artifacts from one layer to another. Examples are the consistent handling of exceptions, consistent usability handling like internationalization, or the combination of concurrent parts of applications. Usually, there are experiences (‘best practices’) from former projects how these aspects should be handled. Traditional methodologies assume skillful modelers which are able to perform this transfer of knowledge. Going

¹AJAX may be used to avoid the default behavior of the browser, but if Javascript is turned off, the application will not work at all.
towards optimizing processes in terms of SPICE requires an explicit handling of this knowledge.

Figure 4: Model Space Restrictions during Co-Design

Figure 4 shows the space of possible specifications on an arbitrary layer of abstraction during a Co-Design process. The modelling languages determine the classes of local models for the four aspects of structure, functionality, interactivity, and distribution. The requirements given from the work products of the previous level as well as the skills of the modelers and developers restrict the class of usable models. A further restriction is given by the technical and organizational environment of the intended application. The resulting class of models determines the modelling alternatives for this layer. Each one is a possible solution to the design problem. Some may be considered to be ‘good solutions’ according to informally given, non-functional parameters.

The use of SPICE in this context encourages to extend every Co-Design based development process in the following fashion to move the outcome of the process towards the 'best practice':

(1) The development process starts on a certain layer of abstraction and climbs down to the implementation layer. If the system is developed from scratch, development starts with the application domain description, otherwise a less abstract layer is chosen.

(2) The development activities are executed on each layer. Development on a certain layer can be considered as a small development process in SPICE. Thus, this process has to be evaluated and its capabilities have to be determined. Because the client is the only stakeholder who is able to evaluate all functional and especially non-functional parameters, the assessment has to take place in the client’s world. That’s why a prototype of the application has to be created. Development of a prototype is a process itself which consumes valuable resources. That’s why a mechanism for rapid prototyping is needed where prototypes are automatically generated out of the existing specification extended by contextual information representing experiences from former processes.

(3) The capability determination induces a refinement of the outcome of the process. Refinement can be done in two different ways: either manually or by incorporating certain application aspects, e.g., adding transitions to the example from Figure 3 to reflect cancelled
and restarted scenarios. Aspect oriented refinement is only possible if the modelling language supports an appropriate algebra for changing the specification.

(4) If the refined specification has been accepted, it has to be translated to the languages of the subsequent layer (‘model realization’). The realization is done by using and extending previously defined mapping functions to incorporate experiences from former projects. The development subprocess for the next layer starts as soon as the translation process is finished.

(5) After the development process reached the implementation layer, the whole process has to be evaluated to adapt the transformation routines for forthcoming developments.

Figure 5 depicts the flow of work products during this development process. The strategy aims at executable specifications in terms of a Model Driven Software Development: specifications on higher layers of abstraction are automatically transformed to executable programs by a reference machine without manual transformation steps [FCF+06].

Figure 5: Evaluated Development by Rapid Prototyping and Executable Specifications

The problem of executable specifications on high abstraction levels is simple: specifications on high abstraction levels are simply not executable because important parts of the application are not present. For that reason, the application under development is simulated. The simulated prototype consists of two major components: the specification on a certain level of abstraction and a runtime environment that completes the missing facts from the abstract specification.

The specification is linked with the runtime environment. This results in a specification that is complete at the level of implementation and can be directly executed on the target system. If the abstract specification is refined, it replaces parts of the runtime environment. This process lasts until the specification covers all relevant aspects. Beside being the specification’s complement the runtime environment can be used in a second way: creating a production-ready version the runtime environment allows stopping the development process on abstract specification layers.

A golden rule in layered system development is that facts which were modelled at a certain abstraction level are visible in some way in all subsequent specifications. To prevent the
‘forgetful’ design caused by human interception, the design methodology has to be backed by appropriate system support that guides the human developer through this process and enforces the methodology in a formal way but also on a pure pragmatically level. The system provides different kinds of transformation facilities which can be used during the development process and which are updated in the evaluation phase of each development process:

- Transformers for rapid prototyping translate an abstract specification to a specification on the implementation layer such that this specification fulfills the already known requirements as well as all technical and organizational restrictions (it is a valid modelling alternative). Missing parts are filled by defaults, e.g., standard skins for the user interface are generated or data is stored in a database with a derived schema. Transformers are parameterized to allow adaptive prototypes, e.g., different skins for the interface or different data models.

- Transformers for aspect-oriented refinement translate specifications to specifications on the same layer of abstraction such that this specification is a valid modelling alternative for a given aspect of the requirements. For example, multiple scenario specifications may be integrated into a multi-modal application.

- Transformers for model realisations transform valid modelling alternatives on a certain layer of abstraction to valid modelling alternatives on a layer with lower abstraction. If $\mathcal{M}_k$ is a specification on a layer $k$ and $f_{k\rightarrow l}$ is a transformer which translates $\mathcal{M}_k$ to an executable program, then $(\mathcal{M}_k, f_{k\rightarrow l})$ is called an executable specification.

[FCF+06] describes the implementation of a generation framework and runtime environment following this approach. Abstract specifications are represented as XML files. The generation framework was implemented in Java and facilitates a dynamically extensible set of transformation classes which transform the specification’s XML document, e.g., using the DOM API or XSLT transformations for aspect-oriented refinements (e.g., skinning or application of interaction patterns). Java source code is generated for rapid prototyping and model realisations.

5 Conclusion

Web information systems augment classical information systems by modern web technologies. They aim in supporting a wide variety of users with a large diversity of utilization stories, within different environments and with desires for personal web-enhanced work spaces. The development of WIS is therefore adding the user, story and interaction dimension to information systems development. So far information systems development could concentrate on the development of sophisticated structuring and functionality. Distributed work has already partially been supported. Therefore, these systems have been oriented towards a support for work.

Modern development methodologies must be carefully defined. In order to be used by everybody these methodologies must also be managed in the sense that any development step can be compared with the goals of the development process. WIS development adds another difficulty to this process: continuous change. WIS applications typically evolve and change with the attraction of users and application areas, with the fast evolution of supporting technology and with the upscaling of the systems themselves. Therefore, top-down design is
replaced by or integrated with evolutionary design and agile methods. Modern WIS development requires change, revision, reconfiguration on the fly, on demand or on plan.

The Co-Design methodology is one of the methodologies that has been used for the development of classical information systems. Before extending and generalizing the approaches developed for this methodology we carefully assessed the methodology and derived deficiencies of this methodology for WIS engineering. We used an assessment by SPICE that is one of the standards for software process improvement methods. This assessment led to requirements for a fully orchestrated methodology and for development of optimization facilities.

Constant change of systems and continuous change of specifications requires an early execution of any part of the specification. Therefore, we also target in development of supporting technologies that allow to execute changes and to incorporate changes into a running system. The specification is supported by a runtime environment.

References


Quality Assurance in Web Information Systems Development

Klaus-Dieter Schewe, Jane Zhao
Massey University
Information Science Research Centre
Private Bag 11222, Palmerston North
New Zealand
[k.d.schewe|j.zhao]@massey.ac.nz

Bernhard Thalheim
Christian-Albrechts University Kiel
Dept. of Computer Science
Olshausenstr. 40, 24098 Kiel
Germany
thalheim@is.informatik.uni-kiel.de

Abstract

Information Systems that are accessible via the web have become a very important class of large-scale software systems. In order to assure a high quality of such systems we propose a layered approach using integrated methods for all levels. In doing so we bring together knowledge from semiotics, goal-oriented requirements engineering, conceptual modelling, databases, human-computer interaction and web-oriented languages. In this paper we outline our approach and argue why the systematic application of our methodology to web information systems development will result in quality.

1. Introduction

A Web Information System (WIS) is a data-intensive system that uses the world-wide web (WWW) as its primary access channel. Since the invention of the WWW such systems have become ubiquitous. Consequently, as with all large-scale software systems the demand for rigorous quality assurance mechanisms arises naturally.

The first natural idea to assure quality is to provide rigorous software development methods that span from requirements elicitation to systems implementation. For WISs various such methods have been developed such as OOHDM [15], HERA [6], WSDM [4], WebML [3], and our own co-design method [12]. While all methods have their individual merits and all methods address the conceptual modelling of WISs in one or the other way, we believe that co-design addresses far more problems than the others and thus in a sense is more complete. For instance, the method includes a high-level strategic level that is used to capture goals and tasks, sketches requirements for content and functionality, and even specifies the ambience for layout and playout on a very high level of abstraction [9]. This is far more than can be achieved by simple use-cases that form the backbone of requirements engineering in most other approaches. Furthermore, the co-design method pays much attention to WIS usage, which amounts to storyboarding. In a nutshell, a storyboard provides an abstract specification of who will be using the system, in which way and for which goals. Languages and reasoning techniques for storyboarding have been discussed in [12]; pragmatics, i.e. what stories mean for users, has been addressed in [14, 16]. The analogy between storyboarding and scenography as used in theatre and movie production has led to the term “screenography” [8], which combines the strategic ambience with screen layout techniques [17] and scene design, i.e. playout.

In [1] we demonstrated how essential parts of our method can be represented by Abstract State Machines [2] in order to be able to verify desirable system properties, in particular consistency. This is another cornerstone of quality assurance in the co-design approach.

However, consistency verification and the ability to infer parts of a system specification from others cover only a small portion of what we subsume under quality assurance. We should at least also validate that the system captures what the provider expects and what users need. Furthermore, we have to validate that the actual system implementation is an efficient representation of the design, and the probability of errors has been minimised.

We address these additional aspects of quality in this paper. The basis is given by an abstraction layer model [11], which provides the framework for various system models that all serve different purposes. The relationship between layers can be understood as refinement. We address each of the layer starting with the strategic layer that addresses systems requirements, followed by the usage layer that is devoted to storyboarding. In doing so we will emphasise in particular the usage aspect of the WIS. The next lower layer provides a conceptual model of the system, which captures the content and functionality that is to be provided, links them to underlying databases, and offers technics for
system customisation. The presentation layer complements this model with a detailed model of layout and playout. Finally, the implementation layer maps the conceptual and presentation models to an implementation thereby exploiting the wide range of X-technologies that have been developed for web applications, an aspect we do not discuss in this paper. We conclude with a brief summary and outlook.

2. Goal-Driven Requirements Engineering

The strategic layer of a WIS is meant to set the target for the models that are to be developed at the lower layers. The strategic model characterises a WIS by a mission statement describing in general terms what the WIS is about, a utilisation space describing content, functionality and context, a utilisation portfolio describing actors, goals and tasks, and general principles describing the ambience and desired atmosphere of the WIS.

Start from a rough classification scheme for WISs called brand, which takes the form $\text{brand} = \mathcal{P} \mathcal{V} \mathcal{U} \mathcal{A}$. The "provider" $\mathcal{P}$ indicates which role the system plays, thus specifies very roughly what kind of content can be expected from the system. $\mathcal{V}$ stands for “what”, and thus adds more detail to the kind of content offered by the WIS. The "user" $\mathcal{U}$ indicates to whom the services offered by the WIS are directed. The "actions" $\mathcal{A}$ indicate the functionality of the WIS offered to its users. The mission statement complements the brand by an informal, textual description.

While the brand, mission statement will be roughly classified according to roles they have with respect to the WIS. Each role has particular goals, and each of these goals corresponds to a task that is meant to achieve this goal. This does not mean that the task has to be executed by the user in this role; it may well refer to tasks executed by users in other roles. Tasks are broken down into subtasks to a level that elementary tasks can be associated with a single role. In addition, subtasks should refer to subgoals. Furthermore, we obtain dependencies between goals, e.g., being a subgoal, a specialisation, or any other kind of dependency. Thus, we complement the informal description of the system by adding goals.

The relationships between tasks, goals and subgoals can be represented in a graph, which we call a task-goal graph. In these graphs we have three different types of vertices for actors, tasks and goals, respectively. Furthermore, we have five different kinds of edges for task-goal relationships, involvement of an actor in a task, goal-goal-relationships, as well as for subtasks and task specialisation.

While the brand, mission statement, utilisation space and utilisation portfolio aim at the characterisation of content, functionality and usage of the WIS in strategic terms, the ambience characterises how the WIS should be configured. At the end the WIS will be implemented, and presented through web pages, which should convey a uniform impression to the WIS users. Categories characterising the impression of pictures can be used such as energetic, romantic, elegant, refreshing, harmonic, or stimulating. Each of these categories will have implications on the choice of form and colour exploiting knowledge from cognitive psychology. On the strategic level the choice of one of these categories corresponds to the question which impression the WIS shall convey, i.e., which atmosphere is best suited for the envisioned content and functionality.

All activities on the strategic layer are semi-formal. The used methods are brainstorming, communication analysis and linguistic analysis. At the end we obtain a network of content items, goals, tasks, actors, etc. that defines the WIS in very rough terms. In order to assure quality the primary activity will be to check that the list of the network is complete, and no aspect of the desired system has been left out.
Adding considerations regarding the ambience of the WIS at this level further assures a uniform presentation of the system.

3. Syntax, Semantics and Pragmatics in Storyboarding

The business layer of a WIS addresses the usage of the system, which is captured by the storyboard [12]. In a nutshell, a storyboard consists of a story space, which itself consists of a hierarchy of labelled directed graphs called scenarios defining the details of scenes, and a plot that is specified by an assignment-free process, in which the basic actions correspond to the labels of edges in the scenarios, a set of actors, i.e. abstractions of user groups that are defined by roles, which determine obligations and rights, and user profiles, which determine user preferences, and a set of tasks that are associated with goals the users may have. In addition, there are many constraints comprising pre- and postconditions, triggering and enabling events, rights and obligations of roles, preference rules for user types, and other dependencies on the plot.

For all parts of a storyboard formal languages and graphical representations have been defined. Thus, with respect to syntax formal correctness can be easily verified. This includes checking that all goals, tasks, and actors that have been identified on the strategic layer, are captured by the storyboard.

The semantics of the storyboard is defined by the set of stories it permits. A story is a path through the story space. We may consider general stories or stories that are associated with a particular task or role of an actor, which provide different views to the WIS. Here, two problems arise immediately: Detect those stories that are enabled by the constraints, in particular the deontic constraints, and those that are even enforced by these constraints in order to decide which tasks are actually supported by the system, and customise the plot according to user preferences.

Solving the first of these problems involves inferencing with deontic logic [5, 18]. Apart from initial considerations not much research has been undertaken in this direction. For the second problem we exploit the fact that plots can be represented as algebraic expressions in Kleene algebras with tests [7], while user preferences give rise to equations. This suggests a (conditional) term rewriting approach [12, 13] to obtain simplified plots, in which non-preferred stories have been eliminated. As indicated in [13] there are still open problems regarding feasibility and efficiency of the approach.

Besides syntactical correctness and inferencing on the basis of logics used to define the semantics, quality assurance requires the provision of sufficient evidence that the designed storyboard is what the intended users actually expect from the system, in other words: what do the stories mean to the users.

This is where the pragmatics of storyboarding comes in. For this the co-design methodology provides mechanisms for usage and portfolio analysis. Usage analysis is grounded in a detailed understanding of intentions associated with the WIS, which are complemented by life cases, user models and context analysis [14]. Portfolio analysis gives rise to the specification of content and functionality chunks. In the following we concentrate on intentions, life cases, and user models.

The description of intentions is based on a clear understanding of aims and targets of the WIS including a specification of long-range and short-term targets, expected visitors, characteristics of this audience, tasks performed by the users, necessary and unnecessary content, and finally an understanding of restrictions of usage. In general, an intention specifies what one purposes to accomplish or do, i.e. one has in mind to do or bring about. This gives rise to four facets that are based on the general characteristics of WISs: a purpose facet, an intent facet, a time facet, and a representation facet [14].

The analysis of intentions structures the WIS in the sense that among others actors and tasks become apparent. Thus, it already gives rise to a coarse structure of the storyboard. Life cases will complement this by shedding more light into the tasks and the way they are to be executed, i.e. they will give rise to the development of task scenarios. However, life case analysis goes beyond task analysis. It focuses on observations in real life situations, thus takes a user-centered approach. This will help identifying and resolving conflicting or competing intentions, before abstracting and mapping life cases to scenarios within the storyboard.

Life cases are characterized by observations, processes, assessment, individual profiles, inter-personal coherence, significance of time and place, characteristics of users, and experiences and skills. Life case studies are designed to produce a picture of service employment that is as accurate as possible. Determining why, how, where, when and why a service is called using what data provides a better picture for utilisation scenario. As life cases are used for quality control, we must carefully record our observations. We either use a natural language specification or a semi-formal one.

We may extract life cases from observations in reality, which are very useful source, whenever a WIS is going to be developed from scratch or is required to provide a ‘natural’ behaviour. In the latter case users are not required to learn the behaviour of the WIS. Instead, the user can continue using a classical behavioural pattern. Furthermore, interview techniques can be applied to extract life cases.

While life case analysis supports the design of the story space, user models address the actors. In particular, user
models support defining user profiles, which lead to preference rules that are decisive for WIS personalisation [12].

User modelling has changed the development to human-computer interfaces, and allows to tailor systems to the users, their needs, abilities, and preferences. User modelling is based on the specification of the user profile that addresses the characterization of the user, and the specification of the user portfolio that describes the user’s tasks, involvement, and collaboration on the basis of the mission of the WIS.

In general, user profiles can be specified through the education profile based on an insight into the knowledge, skills, and abilities of potential users, the work profile with a specification of the specific work approaches of users, and the personality profile representing the specific properties of users. In our opinion these three profiles cover the most important properties of users that influence the storyboard by means of preference rules associate with them.

4. Database Driven Conceptual Modelling

Once a storyboard has been developed at least partially, the question arises how to support the scenes occurring in it. The conceptual layer provides a model that specifies in an abstract way the content and functionality that is to become available at each scene. For this the co-design methods provides the notion of media type [12].

The fundamental consideration is that whenever an actor enters a certain (elementary) scene, he will be confronted with some data presented to him. He may then select some of these data, enter others in fields provided for this, and select an operation. Depending on the selected and entered data the operation will execute some background actions, and lead the user to another (maybe the same) scene presenting different content to him. An operation may also be a simple navigation that does not require any data to be selected or entered. In addition, we make the assumptions that the data presented to users is extracted out of some underlying database, the background actions update this database, and the operations are refinements of the actions appearing in the story space. Furthermore, we assume that we can apply classification abstraction, so that each supported scene is in fact an instance of some class; consequently instead of data we may exploit data types.

Then the core of a media type can be formalised by a view on some database schema. Which data model is actually used to define an underlying database schema is of minor interest. It is, however, important that the queries that define the views are powerful enough to generate complex values and references, as these are intrinsically present at web interfaces. In [12] a variant of IQL has been used for this purpose. A database over the underlying database schema defines the set of media objects of a certain type, among which are those that are currently in use, i.e. presented to some user.

This notion of view only captures the static, i.e. content part of the conceptual model. In order to capture also functionality, operations are associated with each media type. Such an operation consists of input- and output parameters, a selection type and a body. The input abstracts from data to be entered by a user, the selection type characterises what has to be selected in order to start the operation, and the body specifies what the operation actually does in terms of changes to the database and opening new media objects.

The generalised views that result from adding the operations to the views are further complemented by specifications that cope with different granularity by providing hierarchical versions, and adaptivity to channels, devices and user preferences by specifying cohesion [12].

All the information of the storyboard is mapped onto a single media schema, i.e. a collection of media types. In order to assure quality it has to be checked that this mapping is complete, i.e. no scenes or actions have been left out. Furthermore, the various static, dynamic and deontic constraints that have been specified on the storyboard lead to verification obligations same as the pre- and postconditions and the enabling or triggering events.

In order to prepare for these verification tasks, the syntax and semantics of media types are formally defined. As outlined in [10] the semantics of the operations can be expressed in a higher-order dynamic logic that can be used to formalise proof obligations. However, verification with dynamic logic is a very difficult terrain, and higher-order dynamic logics have not attracted much attention in research so far. Alternatively, checking the various constraints can be incorporated into the operations, which reduces the task to a careful validation that no condition has been left out in combination with the generation of test cases.

5. Screenography for Layout and Playout

The conceptual model developed on the conceptual layer abstracts from presentational aspects. These are added to the media types at the presentation layer. Screenography can be seen as the web analogue of scenography. It extends web application engineering by scenographic and dramaturgic aspects and intends to support the interaction between system and user. Screenography aims at an individualized, decorated playout in consideration of user profiles and portfolios, provider aims, context, equipment and the storyline progress. Heterogeneity of web clients and their different capability provide challenges to screenography.

Defining the atmosphere has already been addressed coarsely at the strategic layer. Due to its definition on a high level of abstraction the atmosphere is independent of equipment features. The reification through page layout
and the sufficiency of the available equipment to play out a specific atmosphere will be checked on the presentation layer. The atmosphere is not only determined by a colour schema, which is only part of the ambience of the presentation. Ambience is also determined by other parameters such as shapes, material and illumination. Nevertheless, visual perception is always affected by the current mood and emotions of a viewer. According to [9] we distinguish ambience-types such as powerful in the sense of dramatic art and vitality, romantic in the sense of romance and passion, balanced in the sense of harmony and balance, etc.

Based on the atmosphere we have to specify layout patterns. Patterns are a powerful conceptual framework for building compelling, effective, and easy-to-use websites [17]. A pattern consists of visual and functional building blocks. According to [9] the functional building blocks realise the access to the presented content and order these. The visual building blocks are important for perception and need to consider the colouring with respect to functionality and aesthetics, the perspective perception of the whole screen, and the visual alignment and partitioning of the screen.

The colouring aspect includes the colour schema development on the basis of the specified atmosphere. Therefore, we have to consider the emotions a user usually associates with colours. The effect of colours can be warm, cool, cold, intensive, hot, light, dark, etc. A user’s perception is also influenced by cultural aspects and age. According to [9] the basis of a colour schema can be a colour chord consisting of n colours (n ∈ {2, 3, 4, 6}) that form a regular polygon in the CMY-based colour circle. The colour chords can be complemented to a quality constrast by changing the saturation.

The visual alignment is based on a tiling of the screen as a two-dimensional surface. The co-design method adopts grids from (conventional) graphic design and used for organising page layouts, e.g. newspapers, magazines and other documents [17]. It also supports more sophisticated tiling approaches such as the Fibonacci grid, which is based on the well-known rhythmic sequence of Fibonacci numbers. If each tile is associated with a colour of a well-chosen colour schema, this enables a desired atmospheric effects as specified in the strategic WIS model [9].

Screenography bases screen and particularly WIS layout on cognitive psychology. The layout of WIS pages contains functional elements such as icons for navigation and functions, and visual elements for presentation of texts based on script and colour and of pictures and structures in different displays. The designed screen is regarded as a structured composition of different elements and is defined as screen layout. Screenography uses three kinds of principles taken from cognitive psychology: principles of visual communication, principles of visual cognition and principles of visual design.

A clear and well-defined design of a screen layout helps to grasp and to understand the content and enables to select and to access the information. It is a precondition for the successful communication. Visual communication is based on the exchange of visual codes with special meanings. Sender and recipient agree on the meaning of the communication utterances which are typically expressions in a visual language adopted and understood by both partners.

A sophisticated specification of visual communication is one precondition for interaction support. It consists of three components: vision, cognition, and processing and memorising characteristics. Vision supports users depending on their physical and physiological properties. Users are used and are limited to certain colouring schemata, to different presentation styles and to different reception styles. Cognition is based on the physiological and psychological abilities and skills of users. We must take into consideration the approach users take while reading content and using functionality. Processing and memorising characteristics are based on the psychological ability to read, integrate, and reason about content provided by a page, and to memorize main parts of the content – these vary a lot among users.

Visual communication starts with separating the visual entities on the screen into elementary layout elements. This separation is an analytical process. Visual elements are compared, processed and memorised after recognising them. The scenario for visual communication can be used for developing the layout, i.e. we reverse the order of visual communication. First, visual elements that should be memorised are developed and integrated with the content and functionality necessary for their recognition. Next, these elements are integrated into a draft of the layout. Finally, adornments, presentations, and placements are added.

The process of transferring experienced vision into analytical vision of visual information is based on specific visual features such as contrast, format, visual analogies, picture dramaturgy, reading direction, visual closeness, and symmetric presentation [8]. We explicitly use these visual features for development of visual elements as well as for their composition.

Principles of visual cognition and visual communication refer to ordering, effect delivery, and visualisation. Ordering is based on an arrangement according to the reading direction and on design according to foreground and background relation. The effect uses background for formation of thematic and optical frame, and schemes of colours and structures. Visualisation is influenced by visual design features such as colour, contrast, composition, overlapping, and cuts.

We can use a number of principles of visual cognition in screenography. Users are limited by their time, attention, scope, and task portfolio. These limitations must be taken
into consideration for layout and playout development. We distinguish four principles that should be preserved: the principle of organisation, the principle of economy, the principle of communication, and screen design standards [8].

We also use a number of principles of visual design in screenography such as the optical vicinity principle, the similarity principle, the closeness principle, the symmetry principle, the conciseness principle, and the reading direction principle [8]. These principles help to organise the elements within a page in a way that correspond to human perception. Elements are always recognised within their context. Their value differs based on their syntax, i.e. the formal and aesthetic value, their semantics, i.e. content and objective value, and their pragmatics, i.e. ethic and applicability value.

6. Conclusion

In this paper we described how quality can be assured in web information systems by applying the co-design approach. In particular, we emphasised the completeness of the method with respect to validation. The approach has been applied in more than 30 large industry projects in Germany covering WISs in diverse areas such as regional information services, e-learning and e-government. As most of the actual pages of these systems are database-generated, the actual maintenance effort has been minimised.

In addition to method completeness we have to assure the correctness of the models and their implementation. For this we already developed verification and inference techniques in [1] and [13]. However, this work is far from being completed. For instance, reasoning about deontic constraints as e.g. in [18] has not yet been taken up, and also the work on personalisation still leaves many open problems that are to be addressed in our future work. Furthermore, the testing of software that is based on X-technologies is a wide area, in which very little research has been conducted so far, leaving many challenges for future research.

References

Capturing Forms in Web Information Systems

Hui Ma¹, René Noack², Faizal Riaz-ud-Din¹, Klaus-Dieter Schewe¹, Bernhard Thalheim²
¹Massey University, Information Science Research Centre
Private Bag 11222, Palmerston North, New Zealand
²Christian-Albrechts University Kiel, Institute of Computer Science
Olshausenstr. 40, 24098 Kiel, Germany
[h.ma|f.riaz-ud-din|k.d.schewe]@massey.ac.nz [noack|thalheim]@is.informatik.uni-kiel.de

Abstract

In Web Information Systems (WISs) forms are one of the major interaction instruments that allow users to enter data and thus direct the communication process with the system. In this paper we approach the capture of forms in a conceptual WIS model. Abstracting from low-level forms processing a form can be considered to be part of an interaction type that can be formalised by a generalised view. In this way the concept of media type, which is central to the co-design approach to WISs, provides a natural way to integrate forms into WIS development.

1. Introduction

A Web Information System (WIS) is a data-intensive system that uses the world-wide web (WWW) as its primary access channel. Many WISs are passive in the sense that users can access them to obtain some information, but interaction with the system is restricted to reading, downloading and navigation. Active WISs allow users in addition to enter data and thus manipulate the system, which reacts to the updates. Looking at the user interface of a WIS, active interaction with the system is mainly supported by forms. A decade ago building form-based interfaces to Information Systems led to various proposals, the work in [4] being a good representative for this.

Therefore, it is natural to ask how forms are captured in WIS development. Some approaches, e.g. those in [14] and [13], are quite explicit about this. Systems are modelled on the level of pages, and then every little activity such as entering data into a field is supposed to be captured. This may work for small web applications, but WISs are usually large systems that require database support, so it is infeasible to use low-level forms processing in WIS development.

Other web development methods such as WebML [1], UML-based methods [2], WSDM [3], HERA [7], OOHDM [11] and our own co-design method [10] take a more abstract, conceptual approach to modelling interaction with a WIS, but surprisingly none refers explicitly to forms. On the other hand the work in [5, 6] presents a specific abstract approach to form-based interfaces that is not linked to any WIS development method.

In this article we want to show that the conceptual model of media types, which forms the core of the co-design method for WIS development [10] actually captures forms in an abstract way. Media types are defined as generalised views. That is, assuming some underlying database schema, a view is defined by a query on that schema. Evaluating it on a database will result in a set of objects, the “media objects”, each of which represents a piece of content presented to a user. The most important extension to this concept of view to turn it into a media type is the association of operations. Then a user can select a portion of the content presented to him, enter additional data, and then start an operation, which will update the underlying database and create new media objects. This approach actually captures the gist of the object action principle in common-user access [8]. Further extensions to media types capture granularity variants and adaptivity to user preferences, access channels and end-devices, which are of minor importance for forms modelling.

We would like to emphasise that our approach has been successfully applied in more than 30 large industry projects in Germany covering WISs in diverse areas such as regional information services, e-learning and e-government. Examples are the online city information system Cottbusnet [12] offering services for tourism, accommodation and event booking, information for citizens, electronic markets, etc., the e-learning system DaMIT [9] for acquiring knowledge about data mining techniques and tools, and the e-government system SeSAM. All these WISs make intensive use of form-based interfaces. As most of the actual pages of these systems are database-generated, the actual maintenance effort has been minimised.
In the following we first demonstrate how to abstract from low-level forms processing. Then we will describe the concept of media type in more detail, before discussing how media types capture forms in an abstract way. We conclude with a brief summary and outlook.

2. Abstraction from Low-Level Form Processing

Let us start with looking at the shape of forms. Usually, we find several fields, in which users can (or must) enter data. These fields can be accompanied by some text or other information, but this is merely needed to let a user know what the field is used for. In case the data to be entered must be taken from a finite list of options, selection menus can be provided, which have to be treated as just a representation choice for a field. Similarly, check-boxes and buttons that allow to make an exclusive or non-exclusive choice between several options, are nothing more than a more user-friendly representation of a particular input field.

2.1. Types

In WISs forms are embedded in web pages, so there is even more surrounding information available. Therefore, a form can be represented by a complex value. A form can be empty or filled-in; both cases differ from each other just by the representing complex value. We can abstract from such complex values using a type system, e.g. (using abstract syntax):

\[
\begin{align*}
t & = \mathbb{I} \mid b \mid (a_1 : t_1, \ldots, a_n : t_n) \mid \{t\} \mid \{t\} \\
& \quad \mid \{t\} \mid (a_1 : t_1 \oplus \cdots \oplus a_n : t_n).
\end{align*}
\]

Here \( b \) represents a not further specified set of base types, each associated with a countable domain, i.e. if \( \mathbb{I} \) is a base type, then \( \text{dom}(\mathbb{I}) = \mathbb{Z} \) specifies that the associated domain is the set of integers. \( \mathbb{I} \) is a further base type with a trivial domain, i.e. \( \text{dom}(\mathbb{I}) = \{\bot\} \) is a singleton set. Furthermore, (\( \cdot \)), \( \{\cdot\} \), \( \langle\cdot\rangle \), [\( \cdot \)] and \( \oplus \) represent constructors for record, finite set, multiset (or bag), list, and disjoint union types. When using these constructors, we use label names \( a, a_i \) in them to distinguish the various components.

We may use these constructors to create other types. Then domains are defined in the usual way:

- \( \text{dom}(\{a_1 : t_1, \ldots, a_n : t_n\}) = \{(a_1 : v_1, \ldots, a_n : v_n) \mid v_i \in \text{dom}(t_i) \text{ for } i = 1, \ldots, n\} \) is the set of records with labels \( a_1, \ldots, a_n \) and values of the coordinates taken from the domains of \( t_1, \ldots, t_n \), respectively.
- \( \text{dom}(\{t\}) = \{v_1, \ldots, v_n \mid n \in \mathbb{N}, v_i \in \text{dom}(t) \text{ for } i = 1, \ldots, n\} \) is the set of finite sets with elements in \( \text{dom}(t) \).

\[ \text{dom}(\{t\}) = \{\{v_1, \ldots, v_n\} \mid n \in \mathbb{N}, v_i \in \text{dom}(t) \text{ for } i = 1, \ldots, n\} \text{ is the set of finite multisets with elements in } \text{dom}(t). \]

\[ \text{dom}(\{t\}) = \{\{v_1, \ldots, v_n\} \mid n \in \mathbb{N}, v_i \in \text{dom}(t) \text{ for } i = 1, \ldots, n\} \text{ is the set of finite ordered lists with elements in } \text{dom}(t). \]

\[ \text{dom}(\{t_1 \oplus \cdots \oplus t_n\}) = \{(a_1 : v_1, \ldots, a_n : v_n) \mid v_i \in \text{dom}(t_i) \text{ for } i = 1, \ldots, n\} \text{ is the disjoint union of the domains } \text{dom}(t_i). \]

For instance, a type \( \mathbb{bool} : \{\mathbb{I}\} \) would represent standard Boolean values. The only values \( \{\bot\} \) and \( \emptyset \) in its domain correspond to true and false, respectively.

As another example, the type \( (\text{red} : \mathbb{I}) \oplus (\text{green} : \mathbb{I}) \oplus (\text{blue} : \mathbb{I}) \oplus (\text{yellow} : \mathbb{I}) \) has just four values in its domain: \( \{\text{red} : \bot\}, \{\text{green} : \bot\}, \{\text{blue} : \bot\}, \{\text{yellow} : \bot\} \). It can therefore be considered as an enumeration type, and its domain can be identified with the set of colours \{\text{red}, \text{green}, \text{blue}, \text{yellow}\}.

Example 1 Take the familiar example of a registration form for a conference, in which we would have to fill in first name, last name, affiliation, postal address, phone, fax, email, plus a choice between regular and student participant, a choice between presenter (in which case the paper numbers have to be entered) and non-presenter, the number of additional proceedings volumes, and the number of additional banquet tickets. This leads to the following type for the registration data:

\[ \text{first}\text{name} : (\text{filled}\text{in} : \text{STRING}) \oplus (\text{empty} : \mathbb{I}), \]
\[ \text{last}\text{name} : (\text{filled}\text{in} : \text{STRING}) \oplus (\text{empty} : \mathbb{I}), \]
\[ \text{affiliation} : (\text{filled}\text{in} : \text{STRING}) \oplus (\text{empty} : \mathbb{I}), \]
\[ \text{address} : (\text{filled}\text{in} : \text{STRING}) \oplus (\text{empty} : \mathbb{I}), \]
\[ \text{phone} : (\text{filled}\text{in} : \text{PHONE}) \oplus (\text{empty} : \mathbb{I}), \]
\[ \text{fax} : (\text{filled}\text{in} : \text{PHONE}) \oplus (\text{empty} : \mathbb{I}), \]
\[ \text{email} : (\text{filled}\text{in} : \text{EMAIL}) \oplus (\text{empty} : \mathbb{I}), \]
\[ \text{student} : (\text{filled}\text{in} : \{\mathbb{I}\}) \oplus (\text{empty} : \mathbb{I}), \]
\[ \text{papers} : \{\mathbb{I}\}, \]
\[ \text{proceedings} : (\text{filled}\text{in} : \text{NAT}) \oplus (\text{empty} : \mathbb{I}), \]
\[ \text{tickets} : (\text{filled}\text{in} : \text{NAT}) \oplus (\text{empty} : \mathbb{I}) \]

Here, STRING, PHONE, EMAIL and NAT are base types with the obvious meaning. Each component of this record type corresponds to an input field disregarding its presentation, and each field can be empty or filled in. In the case of the papers field the empty field corresponds to the value \( \emptyset \). The form can be presented in various ways, e.g., the ‘student?’ entry could be represented by a checkbox, and the ‘papers’ entry by a list of entry fields.
Low level forms processing would check the correct typing of any input in case of submitting the data, which would be iterated until the input data is correctly typed. We can ignore this iteration and assume that when a filled-in form is submitted, we get a value of the specified type above.

In general, we could treat all data presented at a page as a complex value. Only the parts that can be changed could be represented as a form.

**Example 2** The value of the type in Example 1

\[
\text{first\_name : (empty : } \perp \text{), last\_name : (empty : } \perp \text{), affiliation : (empty : } \perp \text{), address : (empty : } \perp \text{), phone : (empty : } \perp \text{), fax : (empty : } \perp \text{), email : (empty : } \perp \text{), student? : (empty : } \perp \text{), papers : } \emptyset \text{, proceedings? : (empty : } \perp \text{), tickets? : (empty : } \perp \text{)}
\]

represents an empty form as it may be presented to a user, whereas the value

\[
\text{first\_name : (filled\_in : “Bill”), last\_name : (filled\_in : “Miller”), affiliation : (filled\_in : “Clark University”), address : (empty : } \perp \text{), phone : (empty : } \perp \text{), fax : (empty : } \perp \text{), email : (filled\_in : bmiller@hatmail.com), student? : (filled\_in : } \emptyset \text{), papers : } \{35, 71\}, proceedings? : (filled\_in : } \emptyset \text{), tickets? : (filled\_in : } 1} \text{)
\]

represents a filled-in form of a regular participant presenting two papers and requesting one additional banquet ticket. No address, phone nor fax was entered.

So, the first step in abstracting from low-level forms processing within a WIS is to assume a user in presented with a complex value of some type, which he may change and submit. In particular, we can assume that the submitted value has the correct type.

**2.2. Operations**

Furthermore, we may provide the possibility of choosing between several operations that can be performed with the submitted data. Again, how exactly an operation is selected is of no importance. Finally, we may allow a user to highlight some of the input data, which corresponds to selecting a derived value.

**Example 3** Continuing our example of conference registration, a user may fill in data as indicated in Example 2. He may then simply choose an operation “register”. However, the user may already be registered and simply wants to change the number of additional banquet tickets from zero to one, in which he may choose an operation “change data”.

In order to allow a user to enter only the data that really has to be changed, he could simply highlight the ‘tickets?’ field only, and e.g., dispense with re-entering an address or phone/fax number as indicated in Example 2.

Thus, in addition to a type we have to provide a set of operations, each of which would depend on some selection type indicating the parts of the presented data that must be selected. We will formalise this idea in the following sections. In addition, we may foresee additional input to be requested from the user, for which dialogue boxes could be used.

### 3. The Concept of Media Type

In this section we recall the concept of media type, which was coined by us in 1997, then after several modifications was consolidated in [10]. A basic idea underlying this concept is that for large WISs it will not be sufficient to abstract from presentation aspects, but it will also be supported to link data presented to WIS users with underlying databases. Therefore, instead of just having a complex value type we consider a view on some database schema, which leads to the following definition of an interaction type.

#### 3.1. Views and Interaction Types

A view \( V \) on a database schema \( S \) consists of a view schema \( S_V \) and a defining query \( q_V \), which transforms databases over \( S \) into databases over \( S_V \).

An interaction type has a name \( M \) and consists of a content data type \( cont(M) \) with the extension that the place of a base type may be occupied by a pair \( \ell : M' \) with a label \( \ell \) and the name \( M' \) of an interaction type, a defining query \( q_M \) such that \( \{t_M, q_M\} \) defines a view, and a set of operations. Here \( t_M \) is the type arising from \( cont(M) \) by substitution of \( URL \) for all pairs \( \ell : M' \).

Note that this definition requires the presence of a base type \( URL \) in the type system with the obvious meaning that its domain contains all (syntactically) valid url-s. Within media types this type is used for abstract identifiers that identify individual media object, i.e., instances of media types. In particular, these identifiers can also be used to model links.

In principle we could use any query language. However, for our purposes the query language used in the views must be powerful enough to create navigation links, i.e., we must create url-s in the result of a query. Query languages satisfying this property have been discussed intensively in [10].

Finite closed sets \( C \) of interaction types define content schemata. Then a database \( D \) over the underlying database schema \( S \) and the defining queries determine finite sets \( D(M) \) of pairs \( (u, v) \) with URLs \( u \) and values \( v \) of type \( t_M \).
for each $M \in C$. We use the notion pre-site for the extension of $D$ to $C$. The pair $(u, v)$ will be called an interaction object in the pre-site $D$.

A media type is an order-extended interaction type $M$ together with an cohesion preorder $\preceq_M$ (or a set of proximity values) and a set of hierarchical versions $H(M)$. Here we will not define the extensions of ordering, hierarchies and cohesion (see [10] for this), as this is of minor importance for our objective of discussing forms modelling within WISs.

The order-extension is intuitive. Cohesion is used to automatically adapt a media type to situations, where user preferences or channel and device restrictions request the content of an interaction object to be split. The idea is to present the most important information first, then complement step-by-step the rest if desired. Hierarchies are used to allow users to switch between content at various granularity levels. Consequently, a media object arises from an interaction object by deriving different versions, each of which is associated with different partitions of the content.

In the following we ignore these extensions, and thus, by abuse of terminology identify the concepts of interaction object and media object. Then we still have to be more precise about operations on interaction types.

### 3.2. Operations on Media Types

An database operation consists of a signature and a body. The signature consists of an operation name $O$, a set of input-parameter/type pairs $t_i : T_i$ and an output type $T'$. The body is recursively built of the following constructs:

- assignment $x_E := \text{exp}$, where $x$ is a variable representing the content of the type $E$ itself or a local variable (including the output-parameters), and $\text{exp}$ is an expression of the same type as $x_E$.
- local variable declaration \begin{itemize}
  \item \text{let } $x : t$,
  \item \text{skip and fail},
  \item \text{sequencing} $S_1; S_2$ and branching \begin{itemize}
    \item \text{if } $P$ \text{ then } $S_1 \text{ else } S_2 \text{ endif}$,
  \end{itemize}
- operation call $E' := O'(\text{in} : \text{exp}_1', \ldots,\text{exp}_j', \text{out} : x'_1, \ldots, x'_l)$, where $O'$ is an operation on the type $E'$ with compatible signature, and
- non-deterministic selection of values New.$f(x)$, where $f$ is a selector on $E$.

An operation on an interaction type $M$ consists of an operation signature, i.e., name, input parameters and output type, a selection type, and a body which is defined via operations accessing the underlying database.

For a selection type we first require a subtype relationship $t_M \leq t$ to the type derived from the content data type. Subtyping $\leq$ is a syntactically defined relationship between types with the decisive property that $t \leq t'$ canonically defines a subtype function $\pi_M^t : \text{dom}(t) \rightarrow \text{dom}(t')$. For the type system defined in the previous section the subtyping relationship is usually defined as follows:

- $t \leq \emptyset$ holds for all types $t$.
- $(a_1 : t_1, \ldots, a_m : t_m) \leq (a_1 : t_1, \ldots, a_n : t_n)$ holds for $n \leq m$.
- $\{t\} \leq \{t'\}$ holds for $t \leq t'$.
- $\{t\} \leq \{t'\}$ holds for $t \leq t'$.
- $[t] \leq [t']$ holds for $t \leq t'$.

Furthermore, we permit to omit components of type $\emptyset$ within each record type, and due to the presence of labels can assume that order is not important. Then the definition of projection functions $\pi_M^t : \text{dom}(t) \rightarrow \text{dom}(t')$ for $t \leq t'$ is straightforward.

However, with selection types we want to go even further allowing to replace a bulk type, i.e., $\{t\}$, $\langle t \rangle$ or $[t]$ by $t$ to indicate that an element out of a set, multiset or list must be selected. Similarly, we permit a union type $(a_1 : t_1) \cup \cdots \cup (a_m : t_m)$ with $m \leq n$ (and again ignoring order) to indicate that the selected value must have a more specific type.

### 3.3. Media Types Capture Forms

From our discussion above it is quite straightforward to see how media types capture forms. In case we consider a form and nothing but a form, we would define a complex value type $t$ for it, thus capturing the various input fields and abstracting from their representation. The defining view would use a constant query resulting in a single object with the empty form as its only interaction object. Then we would associate operations with the interaction type to capture how the input into the form is to be processed.

**Example 4** Let us continue the conference registration example, in which case the type $t$ in Example 1 becomes the content data type cont$(\text{REGISTRATION})$ of an interaction type REGISTRATION. Note that in this case $t = \langle t_{\text{REGISTRATION}} \rangle$ holds, because no links appear within the content data type. The query $\text{getREGISTRATION}$ is defined as a constant query, i.e., it has the form $v : t$, in which the complex value $v$ of type $t$ represents the empty form as defined in Example 2. Whatever an underlying database looks like, executing the query results in a single interaction object $(u, v)$ with this value and a generated value $u$ of type URL.
Operations associated with this interaction type can be “change data” and “register”. For the latter one the selection type would be

\[
\text{firstName} : \text{(filled in : STRING)}, \text{lastName} : \text{(filled in : STRING)}, \text{affiliation} : \text{(filled in : STRING)}, \text{address} :
\]

\[
\text{(filled in : STRING), phone} : \text{(filled in : PHONE)}, \text{fax} : \text{(filled in : PHONE) } \oplus \text{(empty : []), email} : \text{(filled in : EMAIL)}, \text{student?} : \text{(filled in : [ ]), papers} : \{\text{INT}\}, \text{proceedings?} : \text{(filled in : NAT), tickets?} : \text{(filled in : NAT)}
\]

which indicates that only the fax number is optional, but all other fields must be filled in. For the operation “change data” the selection type could be

\[
\text{pt} : \text{(proceedings? : (filled in : NAT), tickets? : (filled in : NAT))} \oplus \text{((p : (proceedings? : (filled in : NAT))) \oplus (t : (tickets? : (filled in : NAT)))}}
\]

indicating that the number of extra proceedings or the number of additional banquet tickets or both must be highlighted – of course, we could foresee more options for change, but for the sake of brevity we omit these here.

In case a form is embedded in a larger page, the defining view of the interaction will be affected, but the selection type will only capture the form-part.

### 4. Conclusion

In this paper we showed that the concept of media type, which is central to the co-design approach to WIS development, captures forms in an abstract way. A media type is defined by a view on some database schema that is extended by operations and other features that are of minor importance for the problem of capturing forms. In particular, there is no need to address any of the low-level forms processing features such as filling in data and checking them, as these are well-captured by the way operations of media types are defined. We can therefore concentrate on designing content and functionality without bothering how forms can be exploited for user input.

We like to point out that media types are in fact much more powerful than any forms model. They can be used to define query forms as consumption types or answer forms production types. Furthermore, operations on media types enable reasoning about WISs in higher-order dynamic logic.

Furthermore, knowing which input is required by the operations associated with a media type allows a designer to develop a “real” form as part of the implementation of media types. This can be done semi-automatically on the basis of preferences for layout and playout style and the types used in the definition of the operation signatures. In this way low level forms processing becomes a generic system component without any need to be specified.

### References


Context Analysis: Toward Pragmatics of Web Information Systems Design

Hui Ma\textsuperscript{1} Klaus-Dieter Schewe\textsuperscript{1} Bernhard Thalheim\textsuperscript{2}

\textsuperscript{1}Massey University, Department of Information Systems & Information Science Research Centre
Private Bag 11 222, Palmerston North, New Zealand, email: [h.ma/k.d.schewe]@massey.ac.nz
\textsuperscript{2}Christian Albrechts University Kiel, Department of Computer Science
Olsbhausenstr. 40, 24098 Kiel, Germany, email: thalheim@is.informatik.uni-kiel.de

Abstract

On a high level of abstraction a Web Information System (WIS) can be described by a storyboard, which in an abstract way specifies who will be using the system, in which way and for which goals. While syntax and semantics of storyboarding has been well explored, its pragmatics has not. This paper contributes context analysis as a step towards closing this gap. We classify various aspects of contexts related to actors, storyboard, system and time, which make up the context space, then analyse each of these aspects in detail. This is formally support by lifting relations. Finally, we analyse how contexts impact on life cases, user models and the storyboard.

1 Introduction

A Web Information System (WIS) is an information system that can be accessed through the world-wide-web. On a high level of abstraction a WIS can be described by a storyboard (Schewe & Thalheim 2005b), which in an abstract way specifies who will be using the system, in which way and for which goals. In a nutshell, a storyboard consists of three parts:

- a story space, which itself consists of a hierarchy of labelled directed graphs called scenarios, one of which is the main scenario, whereas the others define the details of scenes, i.e. nodes in a higher scenario, and a plot that is specified by an assignment-free process, in which the basic actions correspond to the labels of edges in the scenarios,

- a set of actors, i.e. abstractions of user groups that are defined by roles, which determine obligations and rights, and user profiles, which determine user preferences,

- and a set of tasks that are associated with goals the users may have.

In addition, there are many constraints comprising static, dynamic and deontic constraints for pre- and postconditions, triggering and enabling events, rights and obligations of roles, preference rules for user types, and other dependencies on the plot. Details of storyboarding have been described in (Schewe & Thalheim 2005b). An overview of our method for the design of WIS was presented in (Schewe & Thalheim 2005a).

While syntax and semantics of storyboarding has been well explored, its pragmatics apart from the use of metaphors (Thalheim & Dusterhoft 2000) has not. Pragmatics is part of semiotics, which is concerned with the relationship between signs, semantic concepts and things of reality. This relationship may be pictured by the so-called semiotics triangle. Main branches of semiotics are syntactics, which is concerned with the syntax, i.e. the construction of the language, semantics, which is concerned with the interpretation of the words of the language, and pragmatics, which is concerned with the current use of utterances by the user and context of words for the user. Pragmatics permits the use of a variety of semantics depending on the user, the application and the technical environment. Most languages defined in Computer Science have a well-defined syntax; some of them possess a well-defined semantics; few of them use pragmatics through which the meaning might be different for different users.

Syntactics is often based on a constructive or generative approach: Given an alphabet and an set of constructors, the language is defined as the set of expressions that can be generated by the constructors. Constructions may be defined on the basis of grammatical rules.

Semantics of generative languages can be either defined by meta-linguistic semantics, e.g. used for defining the semantics of predicate logics, by procedural or referential semantics, e.g. operational semantics used for defining the semantics of programming languages, or by convention-based semantics used in linguistics. Semantics is often defined on the basis of a set of relational structures that correspond to the signature of the language.

Pragmatics has to be distinguished from pragmatics. Pragmatics means a practical approach to problems or affairs. According to Webster (Web 1991) pragmatics is the “balance between principles and practical usage”. Here we are concerned with pragmatics, which is based on the behaviour and demands of users, therefore depends on the understanding of users.

The six characteristics of WIS that were discussed in (Schewe & Thalheim 2005b) can be mapped to conceptual structures that are used for storyboard specification:

1. We start with the characteristics used for the strategic layer. Main specification elements used are intention and mission. They are mapped to metaphors, general goals, rhetorical figures, and patterns and grids of web pages discussed later.

2. The scenarios reflect the utilisation by actors, for which we envision a number of stories that correspond to real use. These scenarios may be captured through observation of reality. Story spaces and plots are recorded in various level of
detail through the methods discussed in (Schewe & Thalheim 2005b). The stories are reflected in
the storyboard.

3. Content specification is the basis for the media types, i.e. data types and their functions, which
will be introduced in part III. It combines data specification with user requirements and is re-
flected in the content portfolio.

4. Functionality is provided by the media types as required by the storyboard. Typical standard
functions are navigation, retrieval (search), support functions, and feedback facilities.

5. Context is based on tasks, history, and envi-
ronment. We use the specification of context
for restructuring and functionality enhancement,
which will form the basis of XSL transformations
and the onion approach (Binemann-Zdanowicz, Kaschek, Schewe & Thalheim 2004).

6. Presentation depends on the intention, the
provider, the technical environment available and
the users the WIS is targeting at. Presenta-
tion results in the layout and the playout of the
WIS. Layout requires the development of multi-
media presentations for each page. Playout addi-
tionally requires the development of functionality
that supports visits of users depending on the
story they are currently following to achieve their
goals. Layout and playout integrate the chosen
metaphors; they depend on chosen page patterns
and grids as well as on quality requirements.

Conceptual structures and their association are de-
picted in Figure 1.1. We may separate the syntactics
and pragmatics layers. Arrows are used for repre-
senting part-of- or uses- or relates-associations. For
instance, the story is based on the user and the func-
tions. Information metaphors relate content to infor-
mation.

We use the notions of information and content
in a specific manner. Information as processed by
humans, is carried by data that is perceived or no-
ticed, selected and organized by its receiver, because
of his subjective human interests, originating from
his instincts, feelings, experience, intuition, common
sense, values, beliefs, personal knowledge, or wisdom,
simultaneously processed by his cognitive and men-
tal processes, and seamlessly integrated in his re-
callable knowledge. Content is complex and ready-
to-use data. Content management systems are infor-
mation systems that support extraction, storage and
delivery of complex data. Content may be enhanced

Figure 1.1: The Web Utilization Space Based On the Characteristics of WIS

by concepts that specify the semantic meaning of con-
tent objects and by topics that specify the pragmatic
understanding of users.

Therefore, information is directed towards prag-
macics, whereas content may be considered to high-
light the syntactical dimension. If content is enhanced
by concepts and topics, then users are able to capture
the meaning and the utilisation of the data they re-
cieve. In order to ease perception we use metaphors.
Metaphors may be separated into those that support
perception of information and into those that support
usage or functionality.

Users are reflected by actors that are abstractions
of groups of users. Pragmatics and syntactics share
data and functions. The functionality is provided
through functions and their representations. The web
utilisation space depends on the technical environ-
ment of the user. It is specified through the layout
and the playout. Layout places content on the ba-
is of a data representation and in dependence of the
technical environment. Playout is based on function-
ality and function representations, and depends on
the technical environment.

The information transfer from a user A to a user
B depends on the users A and B, their abilities to
send and to receive the data, to observe the data,
and to interpret the data. Let us formalise this pro-
cess. Let \( s_X \) denote the function user by a user X
for data extraction, transformation, and sending of
data. Let \( r_X \) denote the corresponding function for
data receival and transformation, and let \( o_X \) denote
the filtering or observation function. The data cur-
rently considered by X is denoted by \( D_X \). Finally,
data filtered or observed must be interpreted by the
user X and integrated into the knowledge \( K_X \) a user
X has. Let us denote by \( i_X \) the binary function from
data and knowledge to knowledge. By default, we ex-
tend the function \( i_X \) by the time \( t_X \) of the execution
of the function.

Thus, the data transfer and information reception
(or briefly information transfer) is formally expressed
it by

\[
I_B = i_B(o_B(r_B(s_A(D_A))), K_B, t_X).
\]

Figure 1.2: Dimensions of understanding messages
In addition, time of sending, receiving, observing, and interpreting can be taken into consideration. In this case we extend the above functions by a time argument. The function $s_X$ is executed at moment $t_{X,1}$, $t_{X,2}$ at $t_{X,1}$, and $o_X$ at $t_{O,X}$. We assume $t_{S,X} < t_{R,Y} < t_{S,Y} < t_{R,Y}$ for the time of sending data from $A$ to $B$. The time of a computation $f$ or data consideration $D$ is denoted by $t_f$ or $t_D$, respectively. In this extended case the information transfer is formally expressed by

$$I_B = \text{is}(s_B(r_B(s_A(D_A, t_{S,A}), t_{R,Y}), t_{O,Y}), K_B, t_{B,Y}).$$

The notion of interpretation extends the dimensions of understanding of message displayed in Figure 1.2 to a web communication act that considers senders, receivers, their knowledge and experience. Figure 1.3 displays the multi-layering of communication, the influence of explicit knowledge and experience on the interpretation.

The communication act is specified by

- the communication message with the content or content chunk, the characterisation of the relationship between sender and receiver, the data that are transferred and may lead to information or misinformation, and the presentation,
- the sender, the explicit knowledge the sender may use, and the experience the sender has, and
- the receiver, the explicit knowledge the receiver may use, and the experience the receiver has.

In this paper we approach the analysis of WIS usage as the first important part of storyboarding pragmatics. WIS usage analysis consists of three parts:

1. **Life cases** capture observations of user behaviour in reality. They can be used in a pragmatic way to specify the story space. The work on life cases was reported in a previous publication (Schewe & Thalheim 2007a).

2. **User models** complement life cases by specifying user and actor profiles, and actor portfolios. The actor portfolios are used to get a better understanding of the tasks associated with the WIS. The work on user models was reported in a previous publication (Schewe & Thalheim 2006).

3. **Contexts** complement life cases and user models by characterising the situation in which a user finds him/herself at a certain time in a particular location. We classify various aspects of contexts related to actors, storyboard, system and time, which make up the context space, then analyse each of these aspects in detail. This is formally support by lifting relations.

After a brief overview of the literature in Section 2 we approach the specification of contexts in Section 3. Formal aspects of contexts are then dealt with in Section 5. In Section 6 we conclude with a brief summary and a discussion of open issues.

2 Related Work

Storyboarding and also the preceding strategic modelling of WIS (Moritz, Schewe & Thalheim 2005) are unique to our approach to WIS modelling. Other approaches to WIS engineering such as the object-oriented OO-HDM (Güell, Schwabe & Vilain 2000, Rossi, Schwabe & Lyardet 1999, Schwabe & Rossi 1998), WebML (Ceri, Fraternali, Bongio, Brambilla, Comai & Matera 2003), HERA (Houben, Barna, Frasincar & Vdovjak 2003) and variants of UML (Conallen 2003, Lowe, Henderson-Sellers & Gu 2002) concentrate on providing models of content, navigation and interaction by means of extended views, which in our own work is captured by so-called media types (Schewe & Thalheim 2005b). WSDM (De Troyer & Leune 1998) emphasises the additional need for a mission statement and a high-level description of processes and users in the WIS. Quite often high-level modelling of WISs is subject to UML-based methods, in particular variants of use cases. The rationale underlying our work is that this is far too little to capture strategic and usage issues of WISs.

The integration of goals and soft goals into the information systems development process has been proposed by (Mylopoulos, Fuxman & Giorgini 2000, Giorgini, Mylopoulos, Nicchiarelli & Sebastiani 2002). The detailed distinction of intentions as targets, objects, objectives and aims is based on linguistics (Web 1991). The integration of the temporal dimension is necessary because of the information systems context. The extension by the representational dimensions has been made in order to specify aspects of WISs.

Contextual modelling has found its own research community (Bouquet, Seradini, Brezillon, Benerecetti & Castellani 1999), but very little work has been done to integrate contextual modelling into WIS design. The work in (Mylopoulos & Motshng-Pitrik 1995) uses contexts as an approach to modularise conceptual models. The most advanced attempt to modelling context is the work on contextual information bases (CIB) (Akaishi, Spyratos & Tanaka 2002, Theodorakis, Analyti, Constantopoulos & Spyratos 1998, Theodorakis, Analyti, Constantopoulos & Spyratos 1999). Roughly speaking, a CIB-context associates objects with a name and an optional reference to another CIB-context. Thus, both the name and the reference depend on the usage history. As shown in (Kaschek, Schewe, Thalheim & Zhang 2004) CIB-contexts in a slightly generalised form can be combined with media types to provide a formal model of usage context. For this, instead of only associating merely a name with a location $l$ a location $l$ a complex value, is associated with it.

3 Context Determination

Taking the commonly accepted meaning a context characterises the situation in which a user finds him/herself at a certain time in a particular location. In this sense context is usually defined only statically referring to the content of a database. Only very few attempts have been made so far to consider context of scenarios or stories.

More generally, we consider context as everything that surrounds a utilisation situation of a WIS by a user and can throw light on its meaning. Therefore, context is characterised by interrelated conditions for the existence and occurrence of the utilisation situation such as the external environment, the internal state, location, time, etc. For WISs we need to handle the mental context that is based on the profile of the actor or user, the storyboard context that is based on the story leading to a situation, the data context that is based on the available data, the stakeholder context, and the collaboration context. These different kinds of contexts have an influence on the development of the storyboard and must thus be considered for the development of the WIS.

**Example 3.1** Let us consider a travel information system. It is often desirable to resolve the context of utterances. While booking an airline ticket to London...
3.1 Context Space

When determining context we already know the major life cases we would like to support, the intentions associated with the WIS, the user and actor characterisation on the basis of profiles and portfolios, and the technical environment we are going to use. These restrictions enable a more refined understanding of context within a WIS.

The work in (Moritz et al. 2005) characterises a WIS by six intertwined dimensions, one of which is context. We now relate context to the other dimensions, i.e. to the intentions, the usage, the content, the functionality, and the presentation. As presentation resides on a lower level of abstraction, it does not have an impact on context. Content and functionality will be used for context refinement, which we address later. So, we first concentrate on intention and usage. The user model, the specified set of life cases, and the intention can be used for a disambiguation of the meaning and an injection of context. In doing so we distinguish the following facets of context:

**Actor context:** The WIS is used by actors for a number of tasks in a variety of involvements and well understood collaboration. These actors impose their quality requirements on the WIS usage as described by their security and privacy profiles. They need additional auxiliary data and auxiliary functions. The variability of use is restricted by the actor’s context, which covers the actor’s specific tasks and specific data and function demand, and by chosen involvement, while the profile of actors imposes exceptions. The involvement and collaboration of actors is based on assumptions of social behaviour and restrictions due to organisational decisions. These assumptions and restrictions are components of the actor’s context.

**Storyboard context:** The meaning of content and functionality to users depends on the stories, which are based on scenarios that reflect life cases and the portfolios of users or actors. According to the profile of these users a number of quality requirements such as privacy, security and availability must be satisfied. The actor’s scenario context describes what the actor needs to understand in order to efficiently and effectively solve his/her tasks in the actual portfolio. The actor’s determine the policy for following particular stories.

**System context:** The WIS is developed to support a number of intentions. The purposes and intents lead to a number of decisions on the WIS architecture, the technical environment, and the implementation. The WIS architecture has an impact on its utilisation, which often is only implicit and thus leads to not understandable systems behaviour. The technical environment restricts the user due to restrictions imposed by server, channel and client properties. Adaptation to the current environment is defined as context adaptation to the current channel, to the client infrastructure and to the server load. At the same time a number of legal decisions based on regulations, laws and business rules have been incorporated into the WIS.

**Temporal context:** The utilisation of a scene by an actor depends on his/her history of utilisation. Actors may interrupt and resume their activities at any moment of time. As they may not be interested in repeating all previous actions they have already successfully completed, the temporal context must be taken into account. Due to availability of content and functionality the current utilisation may lead to a different story within the same scenario.

We will discuss these various facets of context in more detail later in this section. This entire information forms the context space, which brings together the storyboard specification and the contextual information. Typical questions that are answered on the basis of the context space are:

- What content is required by the context space?
- What functionality is required by the context space?
- What has to be changed for the life cases, the storyboard, etc., if context is considered?

As outlined above the context space is determined by the actors, the scenarios, the WIS itself, and the time. It leads to a specialisation of the content, structuring and functionality of the scenes.

Context is associated with desirable properties of the WIS such as quality criteria and security and privacy requirements. Quality criteria such as suitability for the users or learnability provide obligations for the WIS development process. Though these criteria are rather fuzzy, they lead directly to a number of implementation obligations that must be fulfilled at later stages, i.e. within the development on the implementation layer.
For instance, learnability means comprehensibility, i.e., the WIS must be easy to use, remember, capture and forecast. This requires clarity of the visual representation, predictability, directness and intuitiveness. These properties allow the user to concentrate on the tasks. The workflows and the discourse structure correspond to the expectations of the users and do not lead to surprising situations. They can be based on metaphors and motives taken from the application domain. In the same way other quality criteria can also be mapped to development obligations.

Other properties that may be associated with context refer to the potential utilisation for other tasks outside the scope of the storyboard. In this case we do not integrate the additional tasks into the storyboard, but instead support these tasks, if this in accordance with our intentions. For instance, we might expect further visits targeting at core concerns of the WIS.

**Example 3.2** Sometimes customers may want to use a WIS for a purpose that does not meet the system's mission statement. For example, a customer may use a banking WIS to learn about the loan business, or a bookshop WIS to learn English. Clearly, the larger the gap between the actual customer's intention and the system's mission statement is, the higher the expected costs will be for supporting such customers. If it can be expected that some customers will interact with the WIS in a 'non-standard' way, a decision has to be made whether to support such intentions or not. This implies a modification of the anticipated information space. It shows that our focus on a business model for context modelling is not always a severe restriction.

### 3.2 Additional Aspects

We may consider three additional context facets:

**Provider context:** Providers are characterised by their mission, intentions, and specific policies. Additionally, terms of business may be added. Vendors need to understand how to run the WIS economically. Typical parts of this context are intentions of the provider, themes of the website, mission or corporate identity of the site, and occasion and purpose of the visits of actors. Thus, providers may require additional content and functionality due to their mission and policy. They may apply their terms of business and may require a specific layout and playlist. Based on this information, the WIS is extended by provider-specific content and functionality. The storyboard may be altered according to the intentions of the provider, and life cases may be extended or partially supported. Provider-based changes to portfolios are typical for WISs in e-government and e-business applications.

**Developer context:** The WIS implementation depends on the capability of the developer. Typically we need to take into account the potential environment, e.g., hard- and software, communication channels, the information systems that are to be incorporated, especially the associated databases, and the programming environment developers use.

**Organisational and social context:** The organisation of task solutions is often already predetermined by the application domain. It follows organisational structures within the institutions involved. We captured a part of these structures already on the basis of the portfolio and modelled it by collaboration. The other pars form the organisational context. Collaboration of partners consists of communication, coordination, and cooperation. Cooperation is based on cooperativity, i.e., the disposition to act in a way that is best helpful for the collaboration partners, taking their intentions, tasks, interests and abilities into account. At the same time, collaboration is established in order to achieve a common goal. Actors choose their actions and organise them such that their chances of success are optimised with respect to the portfolio they are engaged in. Additionally, the social context may be taken into account, which consists of interactive and reactive pressures. Typical social enhancements are socially indicated situations such as welcome greetings, thanking, apologising, and farewell greetings.

### 4 Details of Context Specifications

Let us next take a deeper look into the facets of the context space, i.e., examining actor, storyboard, system and temporal context in more detail.

#### 4.1 Actor Context

The context of an actor is based on his/her intentions. According to the actor’s profile s/he needs support to fulfil the expectations with respect to the quality of information and work. The social and intellectual interests of the actor may also be part of the actor’s context. The actor’s profile may be used for a refinement of the actor’s context leading to the following four specific kinds of context:

**Actor projection context:** Actors may act on their expectations. In this case, they intentionally drop portions of content or functionality and project the current content and functionality to the “normal” case. This projection leads to an implicit context. For instance, within a travel scenario actors are expected to behave like travellers. Another kind of projection is parameter suppression, in which case content or functionality may be dropped or is not noticed whenever it becomes partially irrelevant.

**Actor approximation context:** Often actors need a condensed or approximated information that may be refined later. Typical such approximations are attribute value approximations or structural approximations. For instance, the former ones may allow the WIS to provide first an approximate value for the orientation of the user. A common misuse of approximation is pricing by “starting from”. Structural approximation permits the use of the same symbol for the original object and an abstraction, hence enables the usage of simpler representations.

**Actor ambiguity context:** Sometimes the reference of a symbol can be unambiguous within a narrow scope, in which certain limitations apply, but ambiguous in a larger scope without the limitations. A typical unambiguous symbol is the ‘next’ button in case the next scene lies within the expectations of the actor. Another use of ambiguity can be made by choosing less expressive textual representations. For instance, in a loan application there is no need to clarify that the word ‘bank’ denotes a financial institution.

**Actor mental context:** The mental context captures attitudes and knowledge of actors or other
kinds of alternative states of affairs such as fiction and user expectations. This context is described in terms of proving to real life cases or to expected life cases. Expectations of actors or users can be combined with other more general requirements. The knowledge of the mental context will remain highly incomplete. However, it provides a handle for incorporating users’ and actors’ expectations.

The actor context is intellectual as well as existential. It contributes to enabling the scenarios and the corresponding stories under consideration. The intellectual part is based on the profile of the actor, on habits, traditions, knowledge, experience, etc. It may also be based on the quality requirements an actor is imposing. The actor context restricts the users that might use the WIS, the way the system will be used, and the portfolios. It is based on the intentions for using the system and the portfolio of the actor, e.g. tasks, involvement, and the collaborations the actor is involved in. The existential part is also related to the portfolio under consideration. It is related to the data and functions currently available or provided, and the technical environment.

The specification of this specific actor context becomes necessary whenever we want to support the work of actors that is close to human communication. Human communication exploits the context often to an extreme degree, leaving many things implicit. We do not need a complete decontextualisation as long as the actor can interpret the content and functionality that is provided by the WIS. Contexts provide a mechanism by which we can use the simplest presentation, content and functionality, i.e. the ones that makes the fewest distinctions most of the time while transcending to more expressive presentation, content and functionality only when needed.

Due to these contextual abilities we may restrict presentation, content and functionality to those features that are absolutely necessary. These restrictions may result in presentation principles such as sparse utilisation of additional and not directly necessary content or functionality or economy in utilisation of colours, multimedia objects, and texture.

**Example 4.1** Let us use relocation as an example for an illustration of this principles. An issuer of the relocation life case expects that his personal and identification data are already sufficient for providing him/her all necessary details. So, the context in which the issuer reacts is based on projection and ambiguity context. If we use the information the passport office provides as public information for the city office, then we can adapt the life case directly to the current one. At the same time, the visit of the issuer might be not the first such in his/her life. So, we can now use the information on previous life cases for scaling the life case to the expectations the issuer has. The adaption requires some background knowledge on the handling of life cases in other cities, previous visits, and the profile of the issuer. We may then use a number of questions to figure out, which further adaption or refinement of the life case is applicable. Since some data on the issuer cannot be stored in the system due to regulations and laws we need to repeatedly obtain these data. So, the data we need to capture within the life case are extended by data we need for figuring out which specific life case is under consideration. At the same time we may use this context information for adapting functionality that is provided.

This specific actor context is combined with the portfolio restrictions. Actors with a non-deterministic behaviour do not use high ambiguity or deep projection. At the same time, their mental context and their approximation context must be rather sophisticated. Actors acting more on intention extensively use all four kinds of actor contexts. Task-oriented and reactive behaviour requires support for mental context. Actors acting in collaborations need additional support for their common disambiguation. If actors do not complete their tasks within one session, they need a well-prepared projection context for the case that they resume their tasks. We shall later map these requirements to adaptation rules and control rules for adaptation.

### 4.2 Storyboard Context

Context has also a storyboard dimension. The actor’s context must be combined with the storyboard, life case and portfolio contexts. The latter two selectively condition the situational interest of the actor and the relevance of the current stage for the actor. Based on the relevance we may identify and use properly all the content that should exert the evolution of the current story. We may now use this information for extracting whether a sequence rule, i.e. a rule of the form \( s_1 \rightarrow_2 s_2 \) requiring that a visit of scene \( s_1 \) should be followed by a visit of scene \( s_2 \) can be applied to the current system usage. The rule may hold in general, but is considered to be not applicable if the existence of process \( p_1 \) leading to scene \( s_1 \) does not have a bearing on the existence of process \( p_2 \) leading to scene \( s_2 \). Therefore, the incorporation of context and the derivation of relevance has mainly to do with selecting the best story for the user and is thus used for the adaptation of content and functionality.

The storyboard context can be used for deriving the most appropriate content. We aim at delivering the right data to the right actor with the right tools and scope at the right time. As the storyboard context provides a good source for adaptation of content and functionality to the current stage of the scenario, we can derive the evolution of the life case and add this context information to the context space. This context allows a treatment of the expectations of the actor. Therefore, each scene in a scenario is provided with a pre-scene context, scene context, and a post-scene context.

- **The pre-scene context** consists of all content that has already been delivered to the actor before the appearance at the actual scene. This information can be used to reduce content delivery for scenes. At the same time, this content can be stored in condensed form and made available to the actor when needed, i.e. the actor can revisit the old content whenever this seems to be necessary. Classical browsers only provide a strictly sequential ‘back’ button for this kind of history management. The pre-context of a scene thus contains all valuable content that is collected during a story, and guarantees the availability of this content when needed.

- **The post-scene context** consists of a potential playout of scenes that can be entered after the current scene. If an actor needs some information on the next actions, then this context information can be used. This information is valuable for those actors who intend to drop out of the system. It is also a part of the help information. The post-scene context can be enriched by meta-data describing the content that is provided in the next steps or the data to be produced by the actor. In this case, an intelligent interface may forecast the information need for further steps of the storyboard.
4.3 System Context

The system context is determined by the content and the functions that are provided by the web information system. It consists of at least the following four parts:

Source and acquisition: Source and acquisition is an orthogonal dimension of the WIS. A WIS is supported by media objects that belong to media types as we will explore in detail in part III. In a nutshell, a media object is defined by an extended view on some underlying database, which can then serve for provision of content and functionality of an elementary scene. The databases used for the generation of content form the context of the scenario. We may associate with each scenario the subschemata of these databases that are used for generation of consumed data or for integration of data produced by actors in the scenario.

Associated content: The data that are used for consumed and produced information do not exist in isolation. They are usually associated with other data on the basis of integrity constraints or existence constraints, in particular existence constraints are often not explicitly represented as such, but are embedded into the database schemata used. For instance, we usually associate with objects collected in relationship classes those objects collected in the component classes on which they are based. In this case, we assume that objects in component classes of the relationship type co-exist with objects in the relationship class. We need to consider the environment of content that is currently under consideration together with the data that are associated with this content.

Supported functionality: Functions supporting the actions in scenarios are provided by the WIS. These functions have their own control environment. Typical such control mechanisms are logging, concurrency control, and recovery management.

Security: Security concepts describe encipherment and encryption (keys, authentication, signatures, notarisation, routing control, access control, data integrity, and traffic padding) for data exchange.

4.4 Temporal Context

The temporal context appears in a number of variants, e.g. storage time, validity time, display time, user-defined time, transaction time, etc. The temporal context is applicable in a number of combinations.

Sometimes it is necessary to use all of them, but often it is observed that only one variant of this context is necessary.

Versions show the life cycle of the objects under consideration. As scenarios will have their own life cycle we cannot assume that database changes are directly enforced on websites. Moreover, it may be useful to provide the old content as long as an actor continues with the same story. Versions can often be systematically structured by database system phases:

- **The initialization phase** permits the development of objects storing initial information. Integrity constraints are applicable in a limited form.
- **The production phase** is the central phase, which consists of runtime querying, modification, transaction management, etc.
- **The maintenance phase** is used in productive database applications for clarification of soft constraints, maintenance of constraints that have been cut out from runtime maintenance, and changing the structuring and functionality of the entire database system. Maintenance phases are used in data warehouse applications for recharging the data warehouse with actual information.
- **The archiving phase** is used for archiving the content of a database in a form that data relevant for historical information can be easily retrieved. No data modification is permitted; the only modification operation is to load new changes to the archive.

4.5 Context Templates

We may now combine this context information using the following semi-formal template:

```
Context:
  Extension of:
  Actor context:
    Projection context:
    Ambiguity context:
    Mental state context:
    Characterisation:
  Storyboard context:
    Pre-scene context:
    Post-scene context:
  WIS context:
    Source and acquisition:
    Associated content:
    Supported functionality:
  Security:
  Temporal context:
    Versioning:
    Development phase:
  Provider context:
    Developer context:
    Organisational and social context:
  Based On:
```

5 Formal Aspects of Context Modelling

Context evolves for actors, scenarios, systems, and over time. We model the relation between different contexts by lifting relations. Properties that are valid for a certain context may be lifted to another context. This transfer can be based on local model semantics.
For this recall that a context is determined by actor, storyboard, system and temporal contexts. So let \( \mathcal{A} \) denote the set of actors, \( \mathcal{S} \) the set of scenarios, \( \mathcal{W} \) the set of system characteristics, and \( \mathcal{T} \) the set of time units. Then we can take a subset \( \mathcal{C} \subseteq \mathcal{A} \times \mathcal{S} \times \mathcal{W} \times \mathcal{T} \) to represent a set of contexts. Furthermore, we use a family of contexts \( \{ \mathcal{C}_i \mid i \in I \} \) and a family of statement sets (or theories) \( \{ \mathcal{T}_i \mid i \in I \} \) that are associated with these contexts. Of course, the theory \( \mathcal{T}_i \) describes the properties of the context \( \mathcal{C}_i \).

### 5.1 Lifting Relations

On these grounds we may use local models \( M_{ij} \) for each of these statement sets assuming that the models we consider are enumerated by the second index. More precisely, the models \( M_{ij} \) determine the meaning of content drawn from a language \( \mathcal{L} \) for describing content in view of context \( \mathcal{C}_i \). That is, we use a partial mapping \( \Psi : \mathcal{L} \times \mathcal{C} \to \mathcal{M} \), where \( \mathcal{M} \) denotes a set of pre-determined meanings for content in \( \mathcal{L} \).

We may now distinguish the formula \( \alpha \) occurring context \( \mathcal{C}_i \) from the same formula occurring in another context by considering the context index \( i \), i.e. we consider pairs \( (\alpha, i) \). Lifting relations can be modelled by rules of the form

\[
(\alpha_1, i_1) \ldots (\alpha_n, i_n) \varphi \rightarrow (\alpha, i)
\]

stating that the formulae \( (\alpha_1, i_1) \ldots (\alpha_n, i_n) \) can be lifted to \( (\alpha, i) \) under the side condition \( \varphi \). In addition, a compatibility relation among local models is introduced similar to logics that capture possible world semantics. This compatibility relation is used for entailment and satisfiability. This approach allows us to reason locally and then to transfer the knowledge we gained to other contexts. Based on this coarse clarification of basic notation we develop a number of facilities and extend the specification of the WIS:

#### Context space

The context space is defined on the basis of the content \( \mathcal{C} \), scenarios \( \mathcal{S} \) and actors \( \mathcal{A} \). In Example 3.1 we could use information on the travel and on the airline to exclude options that seem to be less likely. The context space of a WIS for a given content-meaning pair \( c, m \) consists of precisely those contexts, under which the particular content will have that particular meaning, i.e.

\[
\mathcal{C}(c, m) = \{(a, s, w, t) \in \mathcal{C} \mid \Psi(c, (a, s, w, t)) = m\}.
\]

Adaptation of content, functionality, and scenarios to the context that is currently available is based on context infusion. Applying transformation rules we change content, functions, and the presentation. Therefore, we use a context specification for the development of enforcement rules. These rules may restrict scenarios to more specific ones, extend or shrink content, and extend or remove functions.

#### Life case extension and specialisation

The general life case specification can often be specialised, if context is explicitly injected. We need both the more general life cases and the contextualised ones. Whenever the WIS is revised or extended, we can return to more general life cases and generate another contextualisation. Typical specialisations concern changes in the life case flow. We may specialise the data consumed by an actor in dependence of the actor’s context. If we know that actors need special auxiliary information or conversely actors become more knowledgeable during the utilisation of the WIS, then we may adapt the data provided for consumption. At the same time, we can specialise the figures according to the given context. In the same way spatial and temporal information provide a basis for refinement of life cases.

Life cases may be extended to requirements that were collected in the context space. The content context may require a more elaborated content to be provided. The supported functionality may require additional functions, content, or a specific presentation. Intentions may be more specific under consideration of context. For instance, if we want to support a certain usage of a WIS that was not originally intended but became important in order to maintain frequent visits, then the original life case is extended by those associated life cases.

#### Development of a context manager

Context is also bound to scenes and thus evolves within a story. We may expect that content enhances context. For this reason, we introduced the pre-scene context. Therefore, a subsystem that manages the context is needed. This context manager uses the lifting rules introduced above for transferring context to context for scenes, collaborating actors, and the WIS as such. The system also supports the rule-based development of logics over time. We cannot require that the rule system is complete, but it must be consistent. A useful property is commutativity, i.e. the results of firing rules does not depend on their order. The context management system enhances the dialogue management system by adapting and specialising the presentation and injecting context into it.

**Example 5.1** A typical context extension to functionality is associated with the problem to avoid that users trap into losing-track situations. Such situations can be detected based on the user’s behaviour, e.g. invoking the help function repeatedly on similar topics, repeatedly positioning on particular locations and performing similar operations on similar data, excessively navigating through information space without invoking any reasonable functionality, looking repeatedly for FAQs on similar topics, attempting to enter a discussion forum, and sending email to the site administrator.

User aid that can be provided for losing-track situations is giving access to a thesaurus of the subsystem the user is accessing. Furthermore, the respective business model may be exposed to the user together with an explanation that is adapted to a particular user type. Similarly, access to a FAQ list suitable for the user and the accessed subsystem may be given. Furthermore, improved search facilities and examples targeting at the subsystem accessed may be provided.

#### 5.2 Adaptivity

The idea of adaptivity is to equip the system with enough additional information and rules that would render it possible to engender the right content and functionality for the current situation. That is, the system is supposed to act according to the dictum ‘you take care of the specification, and the system will take care of itself and adapt to the current use’.

Two content objects \( c_1, c_2 \) are synonymous in the context \( C_i \in \mathcal{C} \) iff \( \psi(c_1, C_i) = \psi(c_2, C_i) \). They are
totally synonymous iff \( \psi(c_1, C_i) = \psi(c_2, C_i) \) holds for all contexts \( C_i \in \mathcal{C} \). They are epistemically synonymous within a scenario \( s \) for an actor \( a \) iff \( \psi(c_1, C_i) = \psi(c_2, C_i) \) holds for all contexts \( C_i \in \mathcal{C} \) associated with \( a \) and \( s \).

Applications often require adaptation of processing context, e.g. to

- actual environments such as client, server, and current communication channel,
- user rights, roles, obligations, and prohibitions,
- content required for the portfolio of the current user,
- the actual user with his/her preferences,
- the level of task completion depending on the user, and
- the user’s completion history.

Consider for instance e-learning or e-government websites discussed in (Moritz et al. 2005) and (Schewe, Thalheim, Binemann-Zdanowicz, Kaschek, Kuss & Tschiedel 2005). Citizens may apply for a primary place of residence. In this case, their passport must be changed; otherwise, no change is required. Citizens with school-age children may have to complete additional documents. Completed documents may be decomposed into a suite of documents due to legal restrictions, e.g. by a data protection act requiring that data for city officials and service offices such as the unemployment agency must be separated.

Depending on the role of users, story completion may be scheduled sequentially for some users or in parallel for others. For instance, clerks in a city office may consider documents in parallel, while citizens complete their documents in a sequential mode.

**Example 5.2** Adaptivity may be required at runtime. For instance, people with foreign citizenship may be required to apply for a residence permit. Users may require a varying support depending on the environment that is used for the completion of documents. Users should be supported whenever they are interrupted during task completion.

These requirements lead directly to the requirement to develop a facility for *mutable, adaptable scenarios* for different users, portfolios, and contexts. We shall return to this requirement after introducing templates in the next section. It is our target to develop generic scenarios that can be refined to scenarios by injecting context. This approach is more widely used for WISs than one would expect. For instance, almost all information sites of cities and regions provide a very similar hotel or event search. The reason is not the existence of a development monopoly but rather the evolution of these search facilities to standards. These standards are not officially agreed, but have been formed by copying successful solutions.

6 Conclusion

In this paper we approached the pragmatics of Web Information Systems (WIS) design focusing on the method of storyboarding that is an integral part of the codesign approach to WIS design (Schewe & Thalheim 2005a, Schewe & Thalheim 2005b). A storyboard specifies in an abstract way who will be using the WIS, in which way, and for which goals. Thus, the specification of a storyboard captures the navigation paths, i.e. the stories through the “scenes” of the WIS, the action scheme associated with the stories, the actors appearing in the scenes, and the tasks the actors accomplish. In addition, there are various static, dynamic and deontic constraints governing the storyboards.

While syntax and semantics of storyboarding have been well explored, its pragmatics has not. While many methods for WIS design emphasise content modelling, we start from the very fundamental observation grounded in semiotics that content refers to a syntactic dimension, whereas a pragmatic dimension requires dealing with information. This led to the objective to investigate in depth intentions associated with a WIS. The facets of intention arising from this form the basis for our technical development in this paper dealing with life cases, user models, and contexts.

Life cases capture observations in reality, which by means of abstraction can be used to derive scenarios for the storyboard. Integrating these scenarios provides a method for storyboarding. User models are by user and actor profiles, and actor portfolios. The latter ones provide a better understanding of the tasks associated with the WIS. Contexts can be classified according to how they impact on the life cases, the user models, and the storyboard extracted from them.

This work on pragmatics of storyboarding contributes to closing a gap in the codesign methodology for WIS design. It links the formalism of storyboarding to the systems requirements, and provides guidelines and means to derive the complex storyboards from informal ideas about a WIS without any technical bias. So far we have emphasized that work on pragmatics is a decisive part of the methodology, which does not just consist of a collection of formally integrated models, but also has to state how to use them. It would be rather difficult mapping life cases or user models directly to a conceptual model of a WIS, which resides on a much lower level of abstraction as the storyboard. So on the other hand our work emphasizes the need for storyboarding as the decisive tool for high-level WIS engineering. As shown in (Schewe & Thalheim 2007a) this is also the basis for high-level reasoning about WISs addressing such important issues as personalisation of functionality.

Despite the high relevance of pragmatics for the completeness of storyboarding and the codesign methodology as a whole, the work reported in this paper is only part of the story, as it only addresses the context analysis. Together with life cases (Schewe & Thalheim 2006a) and user models (Schewe & Thalheim 2006) they capture usage analysis of WISs, which still does not completely exhaust the problem area associated with pragmatics of storyboarding. We are in the process of writing up a second part of storyboarding pragmatics dealing with WIS portfolios, which combines content and utilisation portfolios that give rise to content and functionality chunks. The content portfolio is used for collecting information requirements. It is based on information needs and demands, and links the storyboard to the lower-level conceptual model of the WIS consisting of a collection of media types. The utilisation portfolio is used for collecting functionality requirements. It describes intentions of users, specific needs and their context.

In addition to this follow-on part on storyboarding pragmatics we are also working on the pragmatism of storyboarding, storyboard refinements, and quality evaluation. All this together plus the ongoing research on logical grounds of storyboarding and their exploitation for reasoning and verification will complete our research on high-level WIS design within the codesign framework.
References


Schewe, K.-D. & Thalheim, B. (2007a), Life cases: An approach to address pragmatics in the design of web information systems, in ‘Proceedings WebIST’07’.


Web (1991), ‘Webster’s ninth new collegiate dictionary’.
Abstract: Utilization of global information resources as a part of risk management is insufficient. The authorities are maintaining information systems mainly for their own purposes, without access to high quality public information sources in Internet and without interoperability between systems of different authorities. Beneficial use of all available information resources would provide an opportunity to create knowledge based on different pieces of information needs powerful distributed knowledge management, mining the information items, analysing the quality of them, and finally creating new information to be utilized. The distributed operation need support of complex network architectures, models supporting mutual understanding over the cultures and language borders, and ability to recognize the context and adapt the results to the new context. This paper opens discussion from different viewpoints to the topic of global risk management. Architectural solutions supporting interoperability, quality of data in wide networks, ubiquity and mobility as well as time dimension of the information space are covered.

1. Introduction

The existing information sources provide a huge amount of information to solve the problems connected to wide catastrophes and disasters. The existing information, however, is

- distributed,
- collected to serve the individual needs,
- provided by different authorities and organisations,
- located sometimes in closed bases.

The information has in many cases also bindings to cultures and contexts, which make the beneficial use of these difficult. Problems are also caused by the low or totally missing interoperability between the systems managing this information, as well as the restrictions for the public use of it.

The existing information infrastructure crosses the geographical borders. Technically the availability of information worldwide is easy and fast. In the same time even the big catastrophes and disasters have common meaning: the consequences are often worldwide, people representing different nationalities are part of it, and the responsibility to recover the damages is common. Even in the smaller and local
accidents the information available from one source would be helpful in solving the problem in another context. Because of that improved information management has high demand in the context of global risk management.

The modes of reactivity in the existing or becoming situation would be classified in the following way. In passive mode the situation is registered only but it does not cause any actions. This kind of situations may provide information to the existing bases and would be beneficial for later use. A good example is passive earthquake registration to the publicly available databases accessible via Internet. In the reactive mode, after registering the situation some (pre-planned) actions are triggered. Most legacy information systems developed to support decision making by authorities are alike. The preactive mode provides support for decisions to prepare the responsibilities for the becoming events in advance. Good example of this kind of system is the mud flow warning system introduced later in this paper, or the systems developed for tsunami recognition. In proactive mode the system provides in advance information and guidelines to impact in the becoming event. This needs, in addition to the preaction, also knowledge how to avoid the becoming event totally or what should be done to decrease the level of damage. The advanced level is the foresight – ability to foresee the becoming events in advance. This is usually based on the complex modelling of the situations, integrating information of different sources, complex calculations, and ability to understand the original context of the information and adapt it in new context.

This paper covers some views to support improved levels of reactivity in the connection of global risk management. One of the main elements is the ability to benefit on the information managed by the different legacy systems and public source in a seamless way; this needs architectural solutions based on open interfaces between systems and improved adaptive modelling of data to be provided by one data source to be used by another one, even over cultures and contexts. This topic will be discussed in chapter 2. The higher understanding of information is based on the advanced knowledge management; this topic is discussed in Chapter 3 by introducing the concept of “Next Generation Web” and by opening the discussion on the role of ubiquity in knowledge processing (Chapter 4). In the case of distributed knowledge management there exist a problem of fast availability of knowledge items and ability to connect these in the utilizable form. A Knowledge Grid Platform for Collaborative Knowledge System is introduced in Chapter 5. This Grid-architecture is developed especially to connect to each other local knowledge management systems to the global knowledge grid. As an example in Chapter 6 a preactive (proactive) risk management system, developed for mud flow warnings is introduced. Chapter 7 opens discussion on the Temporal Information Processing in the Context of Global Risk Management.

This paper is based on the presentations of the panel discussion in EJC 2008 Conference.

2. Towards seamless and mobile systems

In the context of global risk management there exists a growing need for collaboration over the borders of cultures. The collaboration is based on the communication in different forms. Because of that we need adaptive and context aware applications, which are widely available in a seamless way. The SSMC/DDKM (Seamless Services and Mobile connectivity in Distributed Disaster Management) project of Tampere University of Technology (Pori). The vision of the project is specified: “There exists
ways to improve interoperability between legacy systems as well as to provide flexible connectivity of new services to the integrated whole. Beneficial use of mobile devices will have a growing role as instructive (push / by demand) devices and as information providers in the case of disasters. Communication will be supported by models as a joint language – even over cultural borders. Added value (effectivity, quality of data) is available by higher interoperability and by beneficial use the global sources of information (knowledge) in addition to the local ones.”

![Figure 1. The Three-tier architecture supporting interoperability and distributed knowledge processing](image)

Interoperability – especially between the information systems of authorities – is typically low. In addition, the ability of the legacy systems to use external information is often either restricted (for safety reasons) or not used in a beneficial way because of missing standardized interfaces. This is true in spite of the fact that several public sources are able to provide such information that would easily be mined and connected to the existing items of information by using improved distributed knowledge processing technologies. Problems are not technical but cultural and organisational. Technically networking of different applications and devices, as well as the usage of variety of terminals in communication is possible. In SSMC project the concept of “Three Tier Knowledge Management Architecture is specified (Figure 1) applying SOA (Service Oriented Architecture) components is developed. The system components are classified in three categories:

- **General Kernel System (level 0) – GKS**: provides centralized Knowledge Base (KB) and Knowledge Management (KM) services to the next tier nodes and gets processed data from next tier nodes;
- **Local Kernel Systems (level 1) – LKS**: provides local KB and KM services to the next tier nodes, processes data coming from the Sensoring SubSystems (SSS) and provides services to the next tier nodes (SSS, LS);
- **Sensoring and/or Service SubSystems (level 2.1) - SSS**: sensing = raw data production, service e.g. transportation logistics guidelines; SSS produces (and processes) data to the upper level systems, gets data from upper level nodes and processes data / provides specialized services;
• **Legacy Systems (level 2.2) – LS**: provides special data and uses upper level services via standardized interfaces;

• **Client Systems (level 3) – CS**: uses services from different levels of nodes based on fixed or service oriented connection to other nodes.

Utilizing the SOA components encourages transfer from applications towards service brokering served by intelligent agents connecting service requests to services available. More detailed the architecture principles are described by Jaakkola in [7].

The concept of *seamless* availability services covers different viewpoints:

• **Seamless = Invisible**: the user does not have to know the services available; in special situations these can be pushed or opened to be available without special demand of the user. This type of services are important especially to guide people in the disaster area to avoid risks and damages.

• **Seamless = Symmetric**: Services are provided by terminal and location independent way. This can be implemented by using standardized interfaces and data channels.

• **Seamless = By demand**: Services are available if needed (service push, information demand).

• **Seamless = Adaptable**: Best service available is utilized instead of the fixed connections.

In practice it is question on context aware and adaptive services.

The role of *mobile* devices is diverse, because of the fast growing processing capacity and the amount built in properties. Mobile device can be characterized as a sensor, advisor, messenger and its universal usability. The built in properties make it a multi-sensor device: it is location sensitive, able to sensor environmental issues (specialized built in sensors / external ones connected with RF), record live images (camera) as well as able to make user monitoring (e.g. heart rate recording in co-operation with compatible sensors). Mobile infrastructure is growing towards ubiquity – services are communicating with mobile terminals and able to collect context sensitive data, to make context sensitive knowledge analysis and utilization, also to recognize its user and provide selective messaging. In the ubiquitous world the users have become location independent.

The role of *models* is to support cross-cultural communication as a standardized way to specify the phenomenon (situation). Models for that purpose are semi-formal and universal, able to model behaviour (process models), data / information / knowledge, or structures of them. In disaster management context we speak about situational awareness: it is a description / specification of a certain situation. Traditionally these are descriptive and based on literal format, which make them language and culture dependent. By providing easy-to-use tools to translate the literal models – or kernel parts of them – in the form of semi-formal models, we are able to win the culture dependency. In cross cultural model based communication is needed – especially in taking account that the wide and large disasters are usually global and the recovering activities are in many cases implemented in the co-operation with several nations. The *global* dimension of disaster management can be seen also in a way to get beneficial use of the experiences of others. Benchmarking – experiences moved to new contexts may be useful to solve problems in an innovative way transferred in a new context. Even the information sources are globally available: we need means to merge the pieces of information in an intelligent way and create new knowledge. This topic is handled in chapter 3 of this article.
A collection of material handling the topics discussed in this article are available in [8] and in a paper version of the material covering same topics (available in autumn 2008) in the Publication Series of Tampere University of Technology.

3. Intelligent Data Mining and Analysis for Disaster Management

3.1. Conceptual Modelling for Intelligent Data Mining and Analysis

The data mining and analysis task must be enhanced by an explicit treatment of the languages used for concepts and hypotheses, and by an explicit description of knowledge that can be used. The algorithmic solution of the task is based on knowledge on algorithms that are used and on data that are available and that are required for the application of the algorithms. Typically, analysis algorithms are iterative and can run forever. We are interested only in convergent ones and thus need a termination criteria. Therefore, conceptualisation of the data mining and analysis task consists of a detailed description of six main parameters discussed in the following paragraphs.

The data analysis algorithm: A large variety of algorithms has been developed in the past. Each of these algorithms transfers data and some specific parameters of the algorithm to a result. However, algorithms may be restricted in one way or another, e.g., by efficiency or complexity criteria or data quality requirements.

The concept space: The concept space defines the concepts under consideration for analysis. These concepts are modelled within a certain language and can be characterised by certain criteria. Analysis typically target on those criteria that can be either supported or rejected by the data. Furthermore, concepts may be underspecified and be a target of analysis. Concepts may be refined in a variety of ways, e.g. inductively, abductively, by generalisation and classification, by instantiation and by contextualisation.

The data space: The data space typically consists in a multi-layered data set of different granularity. Data sets are typically only small samples compared with all the data that should be considered. We need therefore to know which generalisation, extrapolation and abstraction techniques can be applied to the data. Also the quality of the samples and the unknown (or known) probability distribution of values must be considered. The data space is often describable through certain database schemata. Data sets can be chosen systematically or could be chosen maliciously. Data sets may be enhanced by metadata that characterise the data sets and associate the data sets to other data sets. The data space allows to apply some data exploration techniques such as roll-up or dice operations, querying techniques such as tree queries for separation. Some data sets allow to get information concerning the concept by actively experimenting with it. A very important issue is whether or not the analysis model can handle noisy or erroneous data sources.

The hypothesis space: Generally, an algorithm is supposed to map evidence on the concepts to be supported or rejected into a hypotheses about it. Therefore, one has to choose a set of possible descriptions. Clearly, each criterion contained in the concept space has to possess at least one description in the hypothesis space. However, the hypothesis space may additionally contain descriptions not describing any concept in the concept space. Furthermore, the descriptions provided by the hypothesis space may be slightly different from those ones used in defining the concepts in the concept space.
The prior knowledge space: Here, one has to specify which initial knowledge about the domain the algorithm may use. This generally restricts the analysis uncertainty and/or biases and expectations about the concepts to be analysed. Obviously, specifying the hypothesis space already provides some prior knowledge. In particular, the analysis task starts with the assumption that the target concept is representable in a certain way. Furthermore, prior knowledge may also be provided by “telling” the algorithm that “simple” answers are preferable to more “complex” hypotheses. Finally, looking at important applications one has to take into account that prior knowledge may be “incorrect.” Thus, when developing advanced analysis techniques one has to deal with the problem how to combine or trade-off prior versus new data sets.

The success criteria: Finally, one has to specify the criteria for successful analysis. This part of the specification must cover at least some aspects of our intuitive understanding of analysis. In particular, we have to deal with questions like: “How do we know whether, or how well, the analysis was successful?” “How does the algorithm demonstrate that hypotheses are supported by the concepts and the data?”

![Diagram](image)

**Figure 3. The Knowledge Detection Process of Data Mining and Analysis**

Each instantiation and refinement of the six parameters described above leads to specific data mining task. The result of data mining and data analysis is described within the knowledge space. The data mining and analysis task may thus be considered to be a transformation of data sets, concept sets and hypothesis sets into chunks of knowledge through the application of algorithms. We visualize this process in Figure 3.
3.2. The Kiel Data Mining Workbench:

3.2.1. Towards Quality-Driven Data Mining and Analysis

Intelligent data mining and analysis consists of provision of data at an adequate level of detail and an adequate level of quality and on application of techniques for mining and analysis of data, content, information or knowledge. Knowledge explication additionally requires validation, verification and explanation of the concepts obtained. Data are at an adequate level of details if they are either are abstracted in such a way that they become analysable and are at an appropriate level of quality for the application of algorithms or are concretised in such a way that the phenomena may be discovered. We therefore need a facility to dive into micro- or meso-data or to roll up to macro-data. Macro-data may again be considered to be micro-data within a suite of models. Micro data or raw data or sensor data are analysed according to the quality criteria of data analysis algorithms. Macro-data typically also serve as a facilitating means for explanation of results. Quality-driven data mining and analysis therefore consists of a number of typical interrelated with each other tasks discussed below.

Gathering, gardening of content depends on level of data (macron), meta-data, validity, timeliness, recharging, versions, and quality. We may use approaches that have been developed for data warehouse technology, e.g., for play-in and play-out of data. Algorithmics is a filed of research in computer science that develops patterns of algorithms and clarifies their application to the derivation of specific algorithms. Data mining has resulted in ‘1001’ data mining algorithms. These algorithms may be classified in dependence of quality, of data and of their profile and their portfolio.

Meaningful and reasonable interpretation are necessary before developing the concept space and the hypothesis space and after obtaining analysis results. Otherwise, we lack in explanation facilities of the data mining and analysis results.

Strategies can be developed for further elaboration of data mining and analysis depending on the results that have been obtained so far.

3.2.2. Tool support for Intelligent Data Mining and Analysis

Intelligent data mining and analysis is a complex task that must handle all three aspects of data management: input, storage, and export. We are therefore currently developing a workbench that extends the classical data warehouse architecture by certain tools. Data warehouses use a rigid separation of the three aspects and use thus an architecture with an input, storage and export machine. The architecture is displayed in Figure 3.
The workbench consists of a number of specialised tools such as the following. The **data analysis workbench** provides an intelligent support for data analysis, concept development and hypothesis proliferation. It follows the approach depicted in Figure 2. We are interested in reliable and defeatable analysis results. **The data import and improvement facilities** are based on generic data import forms, support detection of data massives that can be integrated with existing data and are going to provide automatic importers for foreign, census or legal data. **The data export and collaboration facilities** are based on query forms similarly to those developed for ER-based data processing and annotate data with additional metadata such as a citation track. The data warehouse architecture for informed users with survey on available data massives and their usage (conditions, facilities, ...); export interfaces and tools. **Effective access control** - role-, portfolio-, profile- and collaboration-based rights and obligations, and protection against anybody else combined with world-wide exclusive use for partners with citation of usage. **Intelligent integration** of foreign, legacy and new data: transformer for foreign and legacy data; data gardener for consistent data protection and upgrowth.

4. Knowledge Processing in Ubiquitous Computing Environment

Ubiquitous networks provide network universal connectivity. Users can acquire information in the form of digital content anywhere, anytime. Currently, even CPUs on mobile devices have sufficient power to process sound and motion picture data. Multimedia content is already being used widely over various networks.

Nevertheless, the text information, such as web content, blogs, and email, is standard, and most people rely on this information. Consumer-generated media (CGM) provide a wider variety of text with multimedia content. Such information can be used by the general users in their daily lives, and is slowly casting a strong influence on government policies and enterprise management.
In the ubiquitous computing environment, we publish information obtained from ubiquitous devices in the real-world on the Internet. Various network services collect user-generated information. These services sense the information, analyze it, extract knowledge, correlate information (or correlate information with a physical object), create digital contents for each user, and display the content on ubiquitous devices. Users can publish and distribute more information when they browse the generated content on these devices. This circulation of digital content should be controlled by the user's request in real-time, that is, distribution of seamless content in the next-generation Web.

Ambient intelligence is one of the novel information processing technologies. The technology will become invisible, embedded in our natural surroundings, presents whenever we need it. We will operate this by simple and effortless interactions. This information will be attuned to all our senses, adaptive to users, and context-sensitive over multiple devices. Ambient intelligence will be the core technology for Web 3.0. These approaches help in realizing Ambient Intelligence. A complete overview of the same is shown in Figure 4.

In order to realize the above content operation, we must analyze, search and create digital content according to the user's context. Since we often decide our subsequent actions on the basis of the information obtained the in real world, this information should be credible and useful. Furthermore, with the aid of seamless operations, we must enhance the quality of the obtained information.

![Figure 4. Circulation of Knowledge Processing for Ambient Intelligence](image)

We have already developed a functional web architecture, which enables us to acquire digital content from natural surroundings, facilitate seamless searching of more related information on several ubiquitous devices, and edit CGM content automatically and publish information to the Internet [10; 11]
5. Knowledge Grid Platform for Collaborative Knowledge Systems

In emergency management systems, past and future objectives remain the same: “providing relevant communities collaborative knowledge systems to exchange information”. Various communities organize their own knowledge repositories, each of which aggregates perception, skills, training, common sense, and experience of a community of people[21]. Thus, knowledge sharing, searching, analysis and provision are essentially important for realizing knowledge-based modern societies with various knowledge processing facilities in a world of networks. Social and policy issues are to be addressed with the idea of coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations In “Knowledge Cluster Systems” project in National Institute of Information and Communications Technology (NICT), “3-Sites model for Long-Distance Knowledge Sharing, Searching, Analysis and Provision System” is proposed as a core system model consisting of three essential functions distributed in a global area network environment[20] – see figure 5. In this system model, three functional sites are dynamically connected as event-sensing, knowledge analysis, and knowledge provision, respectively, and those sites transmit significant knowledge related to accidental or irregular events from various knowledge resources to actual users. The important feature of this model is to dynamically connect event-sensing, knowledge analysis, and knowledge provision sites, according to occasional contexts in various areas related to accidental or irregular events occurred in global, social and natural environments.

Figure 5. Global knowledge grid model for 3-site knowledge cluster system

Open-access and easy adaptability of emergency management systems will play an increasingly important role. The “global knowledge grid” is an integrated infrastructure of the knowledge cluster systems for coordinating knowledge sharing and problem
solving in distributed environments [22; 23]. The knowledge grid uses the basic functions of a grid and defines a set of additional layers to implement the functions of distributed knowledge discovery, analysis and provision. Those functions are implemented as software modules on the grid nodes in parallel on the basis of a service-oriented architecture (SOA) [16]. The knowledge grid enables collaboration between knowledge providers who must mine data stored in different information sources, and knowledge users who must use a knowledge management system operating on several knowledge bases. Virtual organizations (VOs) form logical groups of the knowledge grid services, in each of which disparate organizations or individuals collaborate in a controlled fashion to achieve a common goal.

The reach of the Internet expands opportunities for public involvement, regardless of physical distance from the disaster area. In the same way as the World Wide Web, the knowledge grid provides a framework of infinitely-evolving knowledge repository by connecting heterogeneous knowledge bases owned by different organizations and communities[15]. A typical example of the connection is based on causal relation from one knowledge base to another knowledge base. For instance, a disaster knowledge base can be connected with healthcare knowledge base by establishing a causal relation in order to find diseases caused by specific disasters. In that way, “a web of knowledge” will be formed as collaboration architectures on demand with the collective intelligence, where collaborative data are treated with social interaction and community management. That is what we call the next generation of World Wide Web or “Web 3.0”.

6. A Mudflow Warning System

6.1. Overview

The mudflow is a dangerous and harmful disaster. It engulfs villages, factories, railways and roads. An important research issue is how to establish a mudflow warning system for sending messages to the people in the dangerous areas when the disaster occurred. In this paper we represent our idea for establishing a mudflow warning system. In the system, monitor cameras are used as sensors for catching the signals of the disaster. The idea is also represented on how to recognize the vision signals of the mudflow and how to broadcast the warning message.

In May 2006, a mudflow engulfed villages, factories, railway and roads in an area nearby Surabaya, the second city of Indonesia. Many people have been driven from their homes by a torrent of hot toxic mud. The people who are late for escaping from the mud torrent lost their precious lives. It is an important issue to establish a mudflow warning system for reducing the damage of the disaster. It is also an important research issue on how to find the mudflow as soon as possible when it flows out from the mouth of the mud volcano and how to broadcast a warning message immediately to the people, who need the information for escaping from the torrent of the hot toxic mud.

When the hot mud flows out from the mouth of the mud volcano, hot stream gas also blows out from the mouth. As the vision signal of the hot stream of the mudflow can be caught by cameras far from the volcano mouth, it is possible to establish a warning system by using monitor cameras as the sensors of the system. When the signals of a mudflow happened far from the monitor cameras are caught and the warning messages are broadcasted immediately, people will be given enough time...
escaping from the dangerous areas. Another advantage of using the monitor cameras is that the monitor cameras can be used to monitor the flowing of the mud. In this paper, a mudflow warning system with monitor cameras as sensors is proposed. The outline of the system and the technique for recognizing the vision signal of the mudflow are represented in Section 6.2 and 6.3.

6.2. The Mudflow Warning System

An experimental system with monitor cameras, a vision signal analyzer and a warning message sender is developed. As shown in Figure 6, in the system, image signals are stored into a video database and transmitted to the vision signal analyzer which analyzes and recognizes the mudflow vision signals.

![Figure 6. The Mudflow Warning System](image)

Positions of the monitoring cameras, e-mail addresses of computers and the positions of the computers are registered in the database of the warning message sender. By using the position information of the monitoring cameras, the position of the volcano mouth of the mudflow can be determined. Based on the position information of computers, the e-mail addresses for broadcasting the warning message can be determined. In our system, cell-phones with GPS are also registered for broadcasting the warning messages to those cell-phones in the dangerous areas.

6.3. Basic idea for recognizing mudflow image signal

In order to recognize the mudflow vision signals, vision features, color and image’s edges and their position information, are derived automatically from the monitoring images in the vision signal analyzer. Independent factors of the vision features are extracted by using a mathematical method referred to as the Singular Value Decomposition. 300 images are used to determine the factors correlated to the mudflow vision signals. Based on our experimental results, the factors correlated to the mudflow vision signals are successfully extracted. More details on the feature deriving and the factor extracting are described in [3].

After the mudflow correlated factors \( \mathbf{F} \) are extracted, a vector space is constructed by the factor. Monitoring images are mapped onto the vector space by using the expanding calculation \( \mathbf{M} \times \mathbf{F} \), where \( \mathbf{M} \) is an image vector and \( \times \) is the vector multiple. Normal of each image vector on the vector space is calculated. When the mudflow vision signals are contained in an image, the normal of the image vector will be greater than a threshold. This characteristic is used for recognizing the mudflow vision signals.
7. Time Dimension of the Information Space

7.1. Temporal Information Processing in the Context of Global Risk Management

Time is an essential dimension of our information space. Temporal information processing (TIPS) has an important role in designing and implementing global risk management applications. In the context of global risk management, temporality can have long and medium term dimensions such as identification of causal relations between disasters and certain diseases. Temporality can also have very time-sensitive, short-term dimensions, for example sharing and updating information about rescue operations. Temporal information processing in the context of global risk management can be studied on four levels: general level, content level, functional level and system level [4].

**General level**: When modelling time, there are two main traditions represented in the literature. One view of time is a set of points without duration. The other is that intervals should be considered as temporal individuals. There are some general time ontologies such as OWL-Time. OWL-Time is ontology of temporal concepts. The ontology provides a vocabulary for expressing facts about topological relations among instants and intervals, together with information about durations, date times and time zones. OWL-Time has been extended to cover temporal aggregates as well. Temporal aggregates are collections of temporal entities. Perdurants, on the other hand, are entities that are only partially present, in the sense that some of their proper temporal parts (e.g., their previous or future phases) may be not present. Perdurants are often known as processes, for example a “rescue operation”. The essential concepts of OWL-Time, temporal aggregates and perdurant ontology are summarized in [5]. Time ontological approaches in the GRM context can be applied to formalize temporal contents of Web resources and to describe temporal properties and functions of Web services.

**Content level**: Natural language-based information systems and knowledge management, which can take advantage of temporal dimensions of information and knowledge, can perform many useful functions. Applications such as temporal information recognition and extraction, question-answering, summarization and visualization can all benefit from analysis and interpretation along the temporal dimensions. In such applications, information and knowledge should be transformed into temporally aware structures that can then be used to solve application-related problems. Temporal mark-up languages are used to transform pieces of information and knowledge into temporally-aware structures [19]. For example TimeML (Markup Language for Temporal and Event Expressions) is a robust specification language for events and temporal expressions in natural language. Other interesting methods for processing temporal semantics of pieces of knowledge are: (a) Allen’s relations between time intervals can be applied to calculate temporal relations between pieces of knowledge [1], (b) topic detection and tracking of news materials can be used for indentifying causal relations between temporal phenomena [14] and (c) temporal data mining is concerned with large sequential data sets for example time series [13]. The knowledge content of a global risk management application requires efficient functions that also include temporal awareness for supporting time-sensitive knowledge sharing, analysis and delivery among remote sites.

**Functional level**: Modelling temporal variations of data and temporal pattern recognition are important issues in global risk management applications. Snapshots
databases only contain current data, which are a snapshot of the current reality. Many application, however need both current and past data and possibly future as well. In the broadest sense a database that maintains past, present and future data is called a temporal database. The activities of rescue organizations, for example the management of rescue operations, are ongoing processes and their information needs and processing capabilities should be considered in a time perspective. That is, to support managerial information needs, as well as others, the relevant knowledge bases should possess a temporal dimension to store, analyze, share and deliver time-varying data. Most data models however do not address issues of maintenance and processing of temporal data. These models either create undue data redundancy and/or provide limited time-processing capacity. These are two possible directions that can be followed for handling temporal data. One is to develop a new model to support time dimension and the other to augment existing data models to support time dimension in a coherent way. [9; 18]. Global risk management requires efficient database models and functions that also include temporal awareness for supporting knowledge sharing, analysis and delivery among remote sites.

**System level:** A Petri net is a formal method for modeling functions and information flows in distributed systems [17]. As a modeling language, it graphically depicts the structure of a distributed system as a directed bipartite graph with annotations. As such, a Petri net has place nodes, transition nodes, and directed arcs connecting places with transitions. The places from which an arc runs to a transition are called the input places of the transition. The places to which arcs run from a transition are called the output places of the transition. In certain cases, the need arises to also model the timing, not only the structure of a model. For these cases, timed Petri nets have evolved, where there are transitions that are timed, and possibly transitions which are not timed. In the context of global risk management, Petri nets and timed Petri nets provide an interesting approach to model a large distributed system and its time-sensitive parts.

To summarize aspects of temporal information processing in the GRM context, we can present three relevant questions for further research: (1) What kind of temporal and time-sensitive knowledge structures can be identified in the GRM-context? (2) What kind of time-sensitive functions and services are needed in the GRM-context? (3) What kind of models and methods we have for temporal information processing in the GRM-context?

### 7.2. Web Continents and Levels of Linking in the Context of Global Risk Management

The World Wide Web does not form a single homogeneous network. Rather, according to [2], it is fragmented and broken into four major continents (Figure 7). Each continent has traffic rules of its own when we want to navigate Web lands. In Central Core each node can be reached from every other node. The nodes of IN Continent are arranged such that following the links eventually brings the user back to Central Core. However, when the user starts from the Core he/she is not allowed to return to the IN Continent. In OUT Continent, all nodes can be reached from the Core. Once the user has arrived OUT, there are no links taking her/him back to the Core. Tubes can connect the IN and OUT Continents. In Tendrils, some nodes attach only to IN and OUT Continents. Nodes of Isolated Islands can not be accessed from the rest of the nodes. They are isolated groups of interlinked resources that are unreachable from the Central Core and do not have links to it.
These four continents significantly limit the Web’s navigability. For example, starting from a node belonging to the Central Core, we can reach all resources belonging to this major continent. IN land and isolated islands cannot be reached from the Core. Is this fragmented structure here to stay? Will the future Web eventually integrate the four continents into one? The answer is simple: As long as the links remain directed, such homogenization will never occur. Search engines can not function effectively.

Advanced linking languages such XML/XLink and its extensions [6] can help us in closed Web applications, such as extranets and intranets, which can be parts of global risk management applications. The situation will change radically when we move from closed applications to open Web environments. XLink type languages, even semantically rich, are not anymore enough. We need to extend the approach. We need context based, semantic calculation of links and knowledge in addition to stable, backbone like, knowledge structures [12]. As shown in Figure 8, we can identify four main levels of linking in the future Web: (a) simple (X)HTML based linking, (b) XML Linking Language (XLink) based level with some link semantics, (c) advanced extensions of XLink with richer link semantics and functionality and finally, (d) context based semantic calculation of links.
8. Summary

The paper is produce in collaboration with the authors, all of them participating the collaborative research projects under the umbrella of “global risk management”. The items handled cover the viewpoints to the solutions supporting seamless interoperability and availability of services, mobile dimension as a part of the information infrastructure, combined ubiquity and mobility, GRID architecture supporting the complex and distributed knowledge processing, and quality of data mined from different sources. In addition the role of time dimension as a part of the information infrastructure was discussed. A concrete proactive risk management application – mud flow control and warning system was introduced. It combines computer vision and knowledge processing in the real disaster situation.

One of the consequences binding the parts of the paper together is the concept of context and the role of modeling. Modelling is seen to be a common language between systems and people – even over the cultures and language borders. However, understanding the context of the information is needed to be able to analyse it in a right way. In addition, ability to adapt the results of analysis and utilize the information in a new context is needed. In conceptual modeling terms “active conceptual modeling” and “context aware adaptive systems” have become important research topics. Simple Google search provides the following results: “Active Conceptual Modeling” 1,090,000 Google fits, “Context aware adaptive system” 1,760,000 Google fits (in June 2008). Conceptual models are not fixed but adaptive based on learning the phenomena of the environment they are used. E.g. in the connection with counter-terrorism it is important to notify, based on the exiting models, the risks approaching. A kind of reference model is used as a prescriptive model to provide opportunity to react in the detected risks. In the case the shape of the risk is changing, the model has to learn about the changes and adapt in the new situation – otherwise it is useless.

The joint activity of the researcher (authors of this paper) is continuing to study and find new technologies, methods and approaches to be applied in connection with the distributed risk management. Main items are modelling technologies, intelligent next generation web environment including ability to mine good quality data taking into account the time dimension, grid architectures supporting complex distributed knowledge processing, and software architectures supporting flexible service based connectivity between systems and services.

References


Databases of Personal Identifiable Information

Sabah S. Al-Fedaghi                                Bernhard Thalheim
Kuwait University - Kuwait                    Kiel University - Germany
sabah@eng.kuniv.edu.kw              thalheim@is.informatik.uni-kiel.de

Abstract

This paper explores the difference between two types of information: personal identifiable information (PII), and non-identifiable information (NII) to argue that security, policy, and technical requirements set PII apart from NII. The paper describes databases of personal identifiable information that are built exclusively for this type of information with their own conceptual scheme, system management, and physical structure.

1. Introduction

This paper explores the privacy-related differences between types of information to argue that security, policy, and technical requirements set personal identifiable information apart from other types of information, which involves establishing a PII database with its own conceptual scheme, system management, and physical structure.

Different types of information of interest in this paper are shown in Figure 1. We will use the term infon to refer to “a piece of information” [4]. The parameters are objects, and the so-called anchors assign these objects such as agents to parameters. Infons can have subinfons that are also infons. Let INF the set of infons in the system.

Four types of infons are defined:
1. So-called “private” information is a subset of INF. “Private” information is partitioned into two types of information: PII and PNI.
2. PII = the set of pieces of personal identifiable information. We use the term pinfon to refer to this special type of infon.
3. PNI = the set of pieces of non-identifiable information.
4. NII = (INF – PII). We use the term ninfon to refer to this special type of infon.

NII is the set of pieces of non-identifiable information and includes all pieces of information except personal identifiable information.

2. Related Works

Separating “private” data from “public” data has already been adopted in privacy-preserving systems. However, these systems do not distinguish explicitly personal identifiable information. The Platform for Privacy Preferences (P3P) is one such system that provides a means for privacy policy specification and exchange but “does not provide any mechanism to ensure that these promises are consistent with the internal data processing” [3]. It is our judgment that “internal data processing” requires recognizing explicitly that “private data” is of two types: personal identifiable information and personal non-identifiable information, and this difficulty is caused by the heterogeneity of data. Hippocratic databases have been introduced as systems that integrate privacy protection within relational database systems [1]. Nevertheless, in principle, a Hippocratic database is a general DBMS with a purpose mechanism. Purposes can be declared for any data item that is not necessarily personal identifiable information.

Figure 1. Types of information..
3. Infons

This section reviews the theory of infons, and simultaneously presents it in the PII context in anticipation of materials (e.g., examples) developed in the remaining part of the paper that introduce a theory of PII infons.

The theory of infons provides a rich algebra of construction operations that can be applied to PII. An infon is a discrete item of information and may be parametric. The parameters represent objects. Anchors assign objects to parameters. PII infons are distinguished by the mandatory presence of at least one proprietor, an object of type identifiable person.

Infons in an application domain such as personal identifiable information are typically interrelated with each other; they partially depend on each other, partially exclude each other, and may be (hierarchically) ordered. Thus we need a theory that allows constructing a “lattice” of infons (and PII infons) that includes basic and complex infons while taking into consideration their structures and relationships. In such a theory, we identify basic infons that cannot be decomposed into more basic infons. This construction mechanism of infons from infons should be supported by an algebra of construction operations. We generally may assume that each infon consists of a number of components. The construction is applied in performing combination, replacement, or removal of some of these components; some of them may be essential (not removable) or auxiliary (optional).

Infons may also be related to each other. We use predicates that relate infons with each other. Since infons may consist of infons, then binary predicates are sufficient to describe relationships among them. Thus, the world of infons can be specified as the triple: (A;O;P) as follows.

- Atomic infons A
- Algebraic operations O for computing complex infons such as combination ⊕ of infons, abstraction ⊤ of infons by projections, quotient □ of infons, renaming of infons, union ⊻ of infons, intersection ⊖ of infons, full negation ¬ of infons, and minimal negation ⫟ of infons within a given context.
- Predicates P stating associations among infons such as the subinfon relation, a statement whether infons can be potentially associated with each other, a statement whether infons cannot be potentially associated with each other, a statement whether infons are potentially compatible with each other, and a statement whether infons are incompatible with each other.

The combination of two infons results in an infon that has all components of the two infons. The abstraction is used for a reduction of components of an infon. The quotient allows concentrating on those components of the first life that do not appear in the second infon. The union takes all components of the two infons and does not combine common components into one component. The full negation allows generating all those components that do not appear in the infon. The minimal negation restricts this negation to some given context.

We require that the subinfon relation is not transitively reflexive. The compatibility and incompatibility predicates are not contradictory. The potential association and its negation must not conflict. The predicates should not span all possible associations among the infons but only those that are meaningful in a given application area. We may assume that two infons are either potentially associated or cannot be associated with each other. The same restriction can be made for compatibility.

This infon world is very general and allows deriving more advanced operations and predicates. If we assume the completeness of compatibility and association predicates we may use expressions defined by the operations and derived predicates. The extraction of application-relevant infons from infons is supported by five operations.

1. Infon projection narrows the infon to those parts (objects or concepts, axioms or invariants relating entities, functions, events, and behaviors) that are of concern for the application relevant infons. For example, a projection operation may produce the set of proprietors from a given infon, e.g., {Mary, John} from John loves Mary.

2. Infon instantiation lifts the general infons to those that are of interest within the solution and instantiates variables by values that are fixed for the given system. For example a PII infon may be instantiated from its anonymized version, e.g., John is sick from Someone is sick.

3. Infon determination is used for selecting those traces or solutions to the problem under inspection that are the most perspective or best fitting for the system envisioned. The determination typically results in a small number of scenarios for the infons that are going to be supported. For example, infon determination to decide whether an infon belongs to a certain piiSphere (PII of a proprietor – to be discussed later).

4. Infon extension is used for adding those facets that are not given by the infon but by the environment or the platforms that might be chosen or that might be used for simplification or support of the infon (e.g., additional data, auxiliary functionality). For example,
infon extension to related non-identifiable information (to be discussed later).

5. Infons are often associated, adjacent, interact, or fit with each other. Infon join is used to combine infons into more complex and combined infons that describe a complex solution. For example, joining atomic PII to form compound PII and a collection of related PII information.

The application of these operations allows extraction of which subinfons, which functionality, which events, and which behavior (e.g., the action/verb in PII) is shared among information spheres (e.g., of proprietors). These shared facilities provide cross-cutting concerns among all information spheres of relevant infons. They also hint at possible architectures of information and database systems and on separation into candidate components. For instance, entity (say, non-person entity) sharing describes which information flow and development can be observed in the information spheres.

The theory of PII infons can be applied in several areas such as the technical and legal aspects of information privacy and security.

4. Personal Identifiable Information

It is typically claimed that what makes the data “private” or “personal” is either specific legislation, e.g., a company must not disclose information about its employees, or individual agreements, e.g., a customer has agreed to an electronic retailer's privacy policy. However, this line of thought blurs the difference between personal identifiable information and other “private” or “personal” information. Personal identifiable information has an “objective” definition in the sense that it is independent of such authorities as legislation or agreement.

PII infons involve relationships (e.g., possession) with their proprietors, persons but non-proprietors, and non-persons such as institutions, agencies, or companies: For example, a person may possess PII of another person, or a company may have the PII of someone in its database. However, proprietorship of PII is reserved only to its proprietor regardless of who possesses copy of it.

To base personal identifiable information on firmer ground, we turn to stating some principles related to such information. For us, personal identifiable information (pinfon) is any information that has referent(s) to uniquely identifiable persons. In logic, reference is the relation of a word (logical name) to a thing.

Following Devin’s formalism [4] an infon has the form \(<R, a_1, \ldots, a_n, 1>>\) and \(<R, a_1, \ldots, a_n, 0>>\). \(R\) is an \(n\)-place relation and \(a_1, \ldots, a_n\), an are objects appropriate for \(R\). 0 and 1 indicate these may be thought objects do, respectively, do not, stand in relation \(R\). For simplicity sake, we may write an infon \(<R, a_1, \ldots, a_n, 1/0>>\) as \(<<<a_1, \ldots, a_n>>\) when \(R\) is known or immaterial.

A PII infon, pinfon, is an infon such that at least one of the objects is a singly identifiable person. Any singly identifiable person in the pinfon is called proprietor. The proprietor is the person about whom the pinfon communicated information. If there is exactly one object of this type, the pinfon is an atomic pinfon; if there is more than one singly identifiable person, it is a compound pinfon. An atomic pinfon is a discrete piece of information about a singly identifiable person. A compound pinfon is a discrete piece of information about several singly identifiable persons. If the infon does not include a singly identifiable person then it called a ninfon.

The following is a series of propositions that establish the foundation of the theory of personal identifiable information. The symbol “→” denotes implication.

1. Inclusivity of INF
\(\sigma \in \text{INF} \leftrightarrow \sigma \in \text{PII} \lor \sigma \in \text{NII}\)

2. Exclusivity of PII and NII
\(\sigma \in \text{INF} \land \sigma \in \text{PII} \rightarrow \sigma \in \text{NII}\)
\(\sigma \in \text{INF} \land \sigma \in \text{NII} \rightarrow \sigma \in \text{PII}\)

3. Identifiability
Let ID denote the set of (basic) pinfons of type \(<<< \text{is}, p, 1>>\) and Let \(\sigma\) be a parameter for a singly identifiable person. Then \(<<< \text{is}, \sigma, 1>>\) → \(<<< \text{is}, \sigma, 1>>\) ∈ \(\text{INF}\)

4. Inclusivity of PII
Let \(n_\sigma\) denote the number of uniquely identified persons in the infon \(\sigma\), then \(\sigma \in \text{INF} \land n_\sigma > 0 \leftrightarrow \sigma \in \text{PII}\)

5. Proprietary
For \(\sigma \in \text{PII}\), let PROP(\(\sigma\)) be the set of proprietors of \(\sigma\). Let PERSONS denote the set of (natural) persons. Then, \(\sigma \in \text{PII} \rightarrow \text{PROP}(\sigma) \in \text{PERSONS}\)

6. Inclusivity of NII
\(\sigma \in \text{INF} \land n_\sigma = 0 \leftrightarrow \sigma \in \text{NII}\)

7. Combination of non-identifiability with identity
Let ID denote the set of (basic) pinfons of type: \(<<< \text{is}, p, 1>>\). Then, \(\sigma_1 \in \text{PII} \leftrightarrow <<<\sigma_2 \in \text{NII} \land (\sigma_3 \in \text{ID})>>\)
assuming \(\sigma_1 \notin \text{ID}\). “=” here denotes the “merging” of two subinfons.

8. Closability of PII
\(\sigma_1 \in \text{PII} \land \sigma_2 \in \text{PII} \rightarrow (\sigma_1 \land \sigma_2) \in \text{PII}\)
9. Combination with non-identifiability
\[ \sigma_1 \in \text{NII } \not\in \sigma_2 \in \text{PII} \rightarrow (\sigma_1 \not\in \sigma_2) \in \text{PII} \]

10. Reducibility to non-identifiability
\[ \sigma_1 \in \text{PII } \not\in \sigma_2 \in \text{ID } \leftrightarrow \sigma_1 \in \text{NII} \]

where \( \sigma_2 \) is a subinfon of \( \sigma_1 \). “\( \not\in \)” denotes removing \( \sigma_2 \).

11. Atomicity
Let \( \text{APII} = \{ \text{atomic personal identifiable information} \} \). Then, \( \sigma \in \text{PII } \land n_\sigma = 1 \leftrightarrow \sigma \in \text{APII} \)

12. Non-atomicity
Let \( \text{CPII} = \{ \text{compound personal identifiable information} \} \). Then, \( \sigma \in \text{PII } \land n_\sigma > 1 \leftrightarrow \sigma \in \text{CPII} \)

13. Reducibility to atomicity
\[ \sigma \in \text{CPII} \leftrightarrow \langle \sigma_1, \sigma_2, \ldots, \sigma_m \rangle, \sigma_i \in \text{APII}, m = n_\sigma, \text{ and } 1 \leq i \leq m \text{ and } \{ \text{PROP(} \sigma_1 \text{)} , \text{PROP(} \sigma_2 \text{)} , \ldots, \text{PROP(} \sigma_m \text{)} \} = \text{PROP(} \sigma \text{)} \]

Next we discuss each of these propositions.

Inclusivity of INF
\[ \sigma \in \text{INF } \leftrightarrow \sigma \in \text{PII } \land \sigma \in \text{NII} \]
That is, infons are the union of pinfons and ninfons. PII is the set of pinfons (pieces of personal identifiable information), and NII is the set of ninfons (pieces of non-identifiable information).

Exclusivity of PII and NII
\[ \sigma \in \text{INF } \land \sigma \in \text{PII } \rightarrow \sigma \in \text{NII} \]
\[ \sigma \in \text{INF } \land \sigma \in \text{NII } \rightarrow \sigma \in \text{PII} \]
That is, every infon is exclusively either pinfon or ninfon.

Identifiability
Let \( \text{b} \) be a parameter for a singly identifiable person, i.e., a specific person, defined as
\[ \text{b} = \text{IND}_1|<< \text{singly identifiable, IND}_1, 1>> \]
where IND indicates the basic type: an individual [4]. That is, \( \text{b} \) is a (restricted) parameter with an anchor for an object of type singly identifiable person. The individual \( \text{IND}_1 \) is of type person defined as
\[ << \text{person, IND}_1, 1>> \]
Put simply, \( \text{b} \) is a reference to a singly identifiable person. We now elaborate on the meaning of “identifiable.”

Consider the set of unique identifiers of persons. Ontologically, the Aristotelian entity/object is a single, specific existence (a particularity) in the world. For us, the identity of an entity is its natural descriptors (e.g., tall, black eyes, male, blood type A, etc.). These descriptors exist in the entity/object. Tallness, whiteness, location, etc. exist as aspects of the existence of the entity. We recognize the human entity from its natural descriptors. Some descriptors form identifiers. A natural identifier is a natural descriptor that facilitates recognizing a person uniquely. Examples of identifiers include fingerprints, faces, and DNA. No two persons have identical natural identifiers. An artificial identifier is a descriptor that is mapped to a natural identifier. Attaching the number 123456 to a particular person is an example of artificial descriptors in the sense that they are not recognizable in the (natural) person. An artificial identifier is a set of descriptors that is mapped to a natural identifier of a person. By implication, no two persons have identical artificial identifiers. If two persons somehow have the same Social Security number, then this Social Security number is not an artificial identifier because it is not mapped uniquely to a natural identifier.

We define identifiers of proprietors as infons. Such definition is reasonable since the mere act of identifying a proprietor is a reference to a unique entity in the information sphere. Hence, \( << \text{is, b, 1>> } \rightarrow << \text{is, b, 1>> } \in \text{INF} \)
That is, every unique identifier of a person is an infon. These infons cannot be decomposed into more basic infons.

Inclusivity of PII
Next we position identifiers as the basic infons in the sphere of PII. The symbol \( n_\sigma \) denotes the number of uniquely identified persons in the infon \( \sigma \). Then we can define PII and NII accordingly:
\[ \sigma \in \text{INF } \land n_\sigma > 0 \leftrightarrow \sigma \in \text{PII} \]
That is, an infon that includes unique identifiers of persons is personal identifiable information. From (3) and (4), any unique personal identifier or piece of information that embeds identifiers is personal identifiable information. Thus, identifiers are the basic PII infons (pinfons) that cannot be decomposed into more basic infons. Furthermore, every complex pinfon includes in its structure at least one basic infon, i.e., identifier. The structure of a complex pinfon is constructed from several components:
- Basic pinfons and ninfons, i.e., the pinfon John S. Smith and the pinfon Someone is sick form the atomic PII (i.e., PII with one proprietor) John S. Smith is sick.
- This pinfon is produced by an instantiation operation that lifts the general infons to pinfons and instantiates the variable (Someone) by a value (John S. Smith).
- Complex pinfons form more complex infons, e.g., John S. Smith and Mary F. Fox are sick

We notice that the operation of projection is not PII-closed since we can define projecting of ninfon from pinfon (removing all identifiers). This operation is typically called anonymization.

Every pinfon refers to its proprietor(s) in the sense that it “leads” to him/her/them as distinguishable entities in the world. This reference is based on his/her/their unique identifier(s). The relationship between persons and their own pinfon is called proprietorship [1]. A pinfon is proprietary PII of its proprietor(s).
Defining pinfon as “information identifiable to the individual” does not mean that the information is “especially sensitive, private, or embarrassing. Rather, it describes a relationship between the information and a person, namely that the information—whether sensitive or trivial—is somehow identifiable to an individual” [5]. However, personal identifiable information (pinfon) is more “valuable” than personal non-identifiable information (ninfon) because it has an intrinsic value as “a human matter,” just as privacy is a human trait.

To exclude such notions as confidentiality that are applicable to the informational privacy of non-natural persons (e.g., companies), the next proposition formalizes that pinfon is applied only to (natural) persons.

For σ ∈ PII, we define PROP(σ) to be the set of proprietors of σ. Notice that |PROP(σ ∈ PII)| = n; Multiple occurrences of identifiers of the same proprietor are counted as a single reference to the proprietor. In our ontology, we categorize things (in the world) as objects (denoted by the set OBJECTS) and non-objects. Objects are divided into (natural) persons (denoted by the set PERSONS) and non-persons. A fundamental proposition in our system is that proprietors are (natural) persons.

**Proprietary**

σ ∈ PII → PROP(σ) ∈ PERSONS

That is, pinfons are pieces of information about persons.

**Inclusivity of NII**

σ ∈ INF ∧ n = 0 ↔ σ ∈ NII

That is, non-identifiable information (ninfon) does not embed any unique identifiers of persons.

**Combination of non-identifiability with identity**

Next we can specify several transformation rules that convert from one type of information to another. These (privacy) rules are important to decide what type of information applies to what operations (e.g., information disclosure rules).

Let ID denote the set of (basic) pinfons of type << is, b, 1>>. That is, ID is the set of identifiers of persons (in the world). We now define construction of complex infons from basic pinfons and non-identifying information. The definition also applies to projecting pinfons from more complex pinfons by removing all or some non-identifying information.

σ1 ∈ PII ↔ <<σ2 ∈ NII ⊠ σ3 ∈ ID >>

assuming σ1 ∈ ID.

That is, non-identifiable information plus a unique personal identifier is personal identifiable information and vice versa. Thus the set of pinfons is closed under operations that remove or add non-identifying information. We assume the empty information ∅ is in NII. “⊔” here denotes “merging” two subinfons. We also assume that there is only a single σ3 ∈ ID added to σ2 ∈ NII; however, the proposition can be generalized to apply to multiple identifiers. An example of proposition 7 is

σ1 = <<John loves apples>> ↔ <<σ2 = Someone loves apples ⊠ σ3 = John>>

Or, in a simpler description: σ1 = John loves apples ↔ {σ2 = Someone loves apples ⊠ σ3 = John}

Proposition can also be applied to the union Ψ of pinfons.

**Closability of PII**

PII is a closed set under different operations (e.g., merge, concatenate, submerge, etc.) that construct complex pinfons from more basic pinfons. Hence,

σ1 ∈ PII ⊠ σ2 ∈ PII → (σ1 ⊠ σ2) ∈ PII

That is, merging personal identifiable information with personal identifiable information produces personal identifiable information. Also, PII is a closed set under different operations (e.g., merge, concatenate, submerge, etc.) that construct complex pinfons by mixing pinfons with non-identifying information.

**Combination with non-identifiability**

σ1 ∈ NII ⊠ σ2 ∈ PII → (σ1 ⊠ σ2) ∈ PII

That is, non-identifying information plus personal identifiable information is personal identifiable information.

**Reducibility to non-identifiability**

Identifiers are the basic pinfons. Removing all identifiers from a pinfon converts it to non-identifying information. Adding identifiers to any piece of non-identifying information converts it to a pinfon,

σ1 ∈ PII ⊠ σ2 ∈ ID ↔ σ1 ∈ NII

where σ2 is a subpinfon of σ1.

Proposition 10 states that personal identifiable information minus a unique personal identifier is non-identifying information and vice versa. “⊠” here denotes removing σ2. We assume that there is a single σ3 ∈ ID embedded in σ1; however, the opposition can be generalized to apply to multiple identifiers such that removing all identifiers produces σ1 ∈ NII.

**Atomicity**

Furthermore, we define atomic and non-atomic (compound) types of pinfons. Let APII = a set of atomic personal identifiable information.

Each piece of atomic personal identifiable information is a special type of pinfon called apinfon. As we will see later, apinfons can be reduced to apinfons, thus simplifying the analysis of PII.
Formally, the set APII is defined as follows. \( \sigma \in PII \land n_\sigma = 1 \leftrightarrow \sigma \in APII \)
That is, an apinfon is a pinfon that has a single human referent. Notice that \( \sigma \) may embed several identifiers of the same person, yet the referent is still one. Notice that apinfons can be basic (a single identifier) or complex (a single identifier plus non-identifiable information).

Non-atomicity
Let CPII = a set of compound personal identifiable information. Each piece of compound personal identifiable information is a special type of pinfon called cpinfon. Formally, the set CPII is defined as follows.
\( \sigma \in PII \land n_\sigma > 1 \leftrightarrow \sigma \in CPII \)
That is, a cpinfon is a pinfon that has more than one human referent. Notice that cpinfons are always complex since they must have at least two apinfons (two identifiers).

The apinfon (atomic personal identifiable information) is the “unit” of personal identifiable information. It includes one identifier and non-identifiable information. We assume that at least some of the non-identifiable information is about the proprietor. In theory this is not necessary. Suppose that an identifier is amended to a random piece of non-identifiable information (noise). In the PII theory the result is (complex) atomic PII. In general, mixing noise with information preserves information.

Reducibility to atomicity
Any cpinfon is privacy-reducible to a set of apinfons (compound personal identifiable information). For example, John and Mary are in love can be privacy-reducible to the apinfons (John and someone are in love and Someone and Mary are in love). Notice that our PII theory is a syntax (structural) based theory. It is obvious that the privacy-reducibility of compound personal identifiable information causes a loss of “semantic equivalence,” since the identities of the referents in the original information are separated. Semantic equivalency here means preserving the totality of information, the pieces of atomic information, and their link.

Privacy reducibility is expressed by the following proposition:
\( \sigma \in CPII \leftrightarrow <<\sigma_1, \sigma_2, \ldots, \sigma_m>>, \sigma_i \in APII, \)
\( m = n_\sigma, \text{ and } 1 \leq i \leq m, \text{ and } \{PROP(\sigma_1), PROP(\sigma_2), \ldots, PROP(\sigma_m)\} = PROP(\sigma). \)
The reduction process produces \( m \) atomic personal identifiable information with \( m \) different proprietors. Notice that the set of resultant apinfons is a compound pinfon. This preserves the totality of the original cpinfon through linking its apinfons together as members of the same set.

5. Categorization of atomic personal identifiable information

In this section, we identify categories of apinfons. Atomic personal identifiable information provides a foundation for structuring pinfons since compound personal identifiable information can be reduced to a set of apinfons. We concentrate on reducing all given personal identifiable information to sets of apinfons. Justification for this will be discussed later.

5.1. Eliminating ninfons embedded in an apinfon

Organizing a database of personal identifiable information requires filtering and simplifying apinfons to more basic apinfons in order to make the structuring of pinfons easier. Proposition (9) tells us that pinfons may carry non-identifiable information, ninfons. This non-identifiable information may be a random noise or information not about the proprietor directly. Removing random noise is certainly an advantage in designing a database. Identifying information that is not about the proprietor clarifies the boundary between PII and NII.

A first concern when analyzing an apinfon is projecting (isolating, factoring) information about any other entities besides the proprietor. Consider the apinfon John’s car is fast. This is information about John and about a car of his. This apinfon can be projected as:
\[ (\text{John’s car is fast}) \Rightarrow \{\text{The car is fast, John has a car}\}, \]
where \( \Rightarrow \) is a production operator.

John’s car is fast information embeds the “pure” apinfon John has a car and the ninfon The car is fast. John has a car is information about a relationship that John has with another object in the world. This last information is an example of what we call self information. Self information (sapinfon = self atomic pinfon) is information about a proprietor, his/her aspects (e.g., tall, short), or his/her relationship with non-human objects in the world. So, it is useful to further reduce apinfons (atomic) to sapinfons (self).

Sapinfon is related to the concept of “what the piece of apinfon is about.” In the theory of aboutness, this question is answered by studying the text structure and assumptions by the source about the receiver (e.g., reader). We formalize aboutness in terms of the procedure ABOUT(\( \sigma \)), which produces the set of entities/objects that \( \sigma \) is “talking” about. In our case, we aim to reduce any self infon \( \sigma \) to \( \sigma’ \) such that ABOUT(\( \sigma \)) \( = \) PROP(\( \sigma’ \)).
Self atomic information represents information about:
- Aspects of proprietor (identification, character, acts, etc.)
- His or her association with non-person “things” (e.g., house, dog, organization, etc.)
- His or her relationships with other persons (e.g. wife, friend, employee, etc.)

With regard to non-objects, of special importance for privacy analysis are aspects of persons that are expressed by sapinfon. Aspects of a person are his (physical) parts, his character, his acts, his conditions, his name, his health, his color, his handwriting, his blood type, his manner, his intelligence, etc. Their existence depends on the person, in contrast to (physical or social) objects associated with him/her such as his/her house, dog, spouse, job, professional associations, etc.

Let SAPII denote the set of sapinfons (self personal identifiable information).

14. Aboutness proposition
\[ \sigma \in \text{SAPII} \iff \text{ABOUT}(\sigma) = \text{PROP}(\sigma) \]
That is, atomic personal identifiable information \( \sigma \) is said to be self personal identifiable information (sapinfon) if its subject is its proprietor. The term “subject” here means what the entity is about when the information is communicated. The mechanism (e.g., manually) that converts APII to SAPII has yet to be investigated.

5.2. Sapinfons involving aspects of proprietor or relationship with non-person

We further simplify sapinfons. Let OPJ(\( \sigma \in \text{SAPII} \)) be the set of objects in \( \sigma \). SAPII is of two types depending on the number of objects embedded in it: singleton, sapinfon and multitude, msapinfon. The set ssapinfons, SSAPII, is defined as:

15. Singleton proposition
\[ \sigma \in \text{SSAPII} \rightarrow \sigma \in \text{SAPII} \land (\text{PROP}(\sigma) = \text{OPJ}(\sigma)) \]
That is, the proprietor of \( \sigma \) is its only object in \( p \).

The set msapinfons, MSAPII, is defined as:

16. Multitude proposition
\[ \sigma \in \text{MSAPII} \rightarrow \sigma \in \text{SAPII} \land (|\text{OPJ}(\sigma)| > 1) \]
That is, \( \sigma \) embeds other objects beside its proprietor.

We also assume logical simplification that eliminates conjunctions and disjunctions of PIIS.

Now we can declare that the sphere of personal identifiable information (piiSphere) for a given proprietor is the database that contains:
1. All sapinfons and msapinfons of the proprietor, including their arrangement in super-infons (e.g., to preserve compound personal identifiable information).
2. Related non-identifiable information to the piiSphere of the proprietor as discussed.

5.3. What is related non-identifiable information?

Consider the msapinfons Alice visited clinic Y. It is msapinfons because it represents a relationship (not aspect of) the proprietor Alice had with an object, the clinic. Information about the clinic may or may not be privacy related information. For example, year of opening, number of beds, and other information about the clinic are not privacy related information. Thus, such information ought not to be included in Alice’s piiSphere. However, when the information is that the clinic is an abortion clinic, then Alice’s piiSphere ought to include this non-identifiable information about the clinic.

6. Justifications for PII databases

We concentrate on what we call PII database, PIIDB, that contains personal identifiable information and information related to it.

**Security requirement**: We can distinguish two types of information security:
(1) Personal identifiable information security, and
(2) Non-identifiable information security.

While the security requirements of NII are concerned with the traditional system characteristics of confidentiality, integrity, and availability, PII security lends itself to unique techniques that pertain only to PII.

The process of protecting PII involves: (1) Protection of the identities of the proprietor, (2) Protection of the non-identity portion of the PII. Of course, all information security tools such as encryption can be applied in this context, yet other methods (e.g., anonymization) utilizing the unique structure of PII as a combination of identities and other information can also be used. Data-mining attacks on PII aim at determining the identity of the proprietor(s) from non-identifiable information; for example, determining the identity of the patient from anonymized information that gives age, sex, and zip code in health records (k-anonymization). Thus, PII lends itself to unique techniques that can be applied in protection of this information.

Another important issue that motivates organizing PII separately is that any intrusion on PII involves information besides the owner’s information (e.g., a company, proprietors, and other third parties, e.g., privacy commissioner). For example, a PII security
system may require immediate alerting of the proprietor of intrusion on his/her PII.

An additional point is that the sensitivity of PII is in general valued more highly than the sensitivity of other types of information. PII is more “valuable” than non-PII because of its privacy aspect, as discussed previously. Such considerations imply a special security status for PII.

**Policy requirements.** Policies that are applied for PII are not applicable to NII (e.g., consent, opt-in/out, proprietor’s identity management, trust, privacy mining). While the NII security requirements are concerned with the traditional system characteristics of confidentiality, integrity, and availability, PII privacy requirements are concerned also with such issues as purpose, privacy compliance, transborder flow of data, third party disclosure, etc. Separating PII from NII can reduce the complex policies required to safeguard sensitive information where multiple rules are applied, depending upon who is accessing the data and what the function is.

In general, PIIDB goes beyond mere protection of data:
1. PIIDB identifies proprietor’s piiSphere and provides security, policy, and tools *to the piiSphere*.
2. PIIDB provides security, policy, and tools *only to proprietor’s piiSphere*, thus conserving privacy efforts.
3. PIIDB identifies inter-piiSphere relationships (proprietors’ relationships with each other) and provides security, policy, and tools to protect the privacy of these relationships.

**7. Personal Identifiable Information Database (PIIDB)**

The central mechanism in PIIDB is an explicit declaration of proprietors in a table that includes unique identifiers of all proprietors in the PIIDB. The principle of uniqueness of proprietor’s identifiers requires that the internal key is mapped one-to-one to the individual’s legal identity or physical location.” This is an important feature in PIIDB to guarantee consistency of information about persons. This “identity” uniquely identifies the piiSphere and distinguishes one piiSphere from another.

PIIDB obeys all propositions defined previously. Some of these propositions can be utilized as privacy rules. As an illustration of the applications of these propositions, consider the case of privacy constraint that prohibits disclosing $\sigma \in$ PII. By proposition (9) above, mixing (e.g., amending, inserting, etc.) $\sigma$ with any other piece of information makes the disclosure constraint apply to the combined piece of information. In this case a general policy is: Applying a protection rule to $\sigma_1 \in$ PII implies applying the same protection to $(\sigma_1 \oplus \sigma_2)$ where $\sigma_2 \notin$ PII.

**8. Conclusion**

The theory of PII infons can provide a theoretical foundation for technical solutions for protection of personal identifiable information. In such approach, privacy rules form an integral part of the design of the system. PII can be identified (hence becomes an object to privacy rules) during processing of information that may mix it with other types of information. We proposed to analyze and process PII as a separate database with clear boundary lines with non-identifiable information. This facilitates meeting the unique requirements of PII.

**9. References**

Abstract-One of the most important aspects of security organization is to establish a framework to identify security-significant points where policies and procedures are declared. The (information) security infrastructure comprises entities, processes, and technology. All are participants in handling information, which is the item that needs to be protected. Our approach is to identify information stream as the principal focus of security consideration, and it is based on recognizing points of transformation in the flow of information as the pivots around which security organization is built. The information stream model is a general, coherent blueprint of security organization without getting into details of specific technology, policies, or practices.

I. INTRODUCTION

Current needs in information security require the integration of policies and user responsiveness to security policies, into addition to technology-based solutions. This is the Information Assurance approach, encompassing aspects of information characteristics, information states, and security measures at the levels of technology, policy and human interfaces. According to Beauregard [3], “IA is not a synonym for computer security; assurance is a much more robust concept that captures the entire process of defending one’s information, information systems, and information processes.”

An information assurance model is one dimension of response to growing demands for integrated security infrastructure requirements within an enterprise. It is a conceptual representation that identifies different levels of various aspects of security without being bogged down with details of specific practice and procedures [10]. Tomhave [10] describes 19 different information security models, but notes that there is only one model can be classified as providing general guidance for achieving such a goal: the McCumber Cube model (see justifications in [8]).

II. THE CUBE MODEL

McCumber’s Information Systems Security model is an abstract framework for the protection of information systems, whether storing, processing, or transiting data, without getting into the details of specific technology, policies, or practices. It has been extended and used to enforce policies across differing types of security-related controls. We reconstruct the model in the context of information stream. The “information states” dimension of the model is defined in terms of 14 stages and 17 passages between stages. It is used for identifying security checkpoints throughout the information stream. Our extended model is based on an information stream that provides a more precise foundation, with fine-grained subprocesses of information handling; thus, it allows tailoring security policies and procedures according to the requirements of each subprocess.

This model aims at providing “an information-centric model that captures the relationship between the disciplines of communications and computer security, without the constraints of organizational or technical changes” [10]. Information is the central focus in this model as money is for economics.

The cube model involves three dimensions, as shown in figure 1. Each dimension includes three aspects:

Figure 1. The McCumber cube model.

1. Information States: These represent the states of (a) transmission, (b) storage and (c) processing.
2. Security Measures: These include the categories of safeguards: (a) technology, (b) policy and procedure, and (c) human factors.
3. Information Characteristics: These make up the known CIA model [5] that describes three significant issues of information security: (a) Confidentiality, (b) Integrity, and (c) Availability.

The cube model identifies 27 types of security-related controls, for example, to audit:
1. The technology-based security controls of information transmission in the categories of CIA
2. The policy-based security controls of information transmission in the categories of CIA
3. The (human factors)-based security controls of information transmission in the categories of CIA
4. The technology-based security controls of information storage in the categories of CIA

... 
27. The (human factors)-based security controls of information processing in the categories of CIA

According to Maconachy et al. [7], the model provides a concise representation of information assurance discipline. It can be used for enforcing policies, independent of technology, across different types of security-related controls.

The cubic model was extended to accommodate the Canadian Trusted Computer Product Evaluation Criteria [9]. Maconachy et al. [7] broadened the scope of information states in the cube model to include transmission characteristics of authentication and Nonrepudiation, as shown in figure 2.

<table>
<thead>
<tr>
<th>Confidentiality</th>
<th>Integrity</th>
<th>Availability</th>
<th>Authentication</th>
<th>Nonrepudiation</th>
</tr>
</thead>
</table>

Authentication concerns the validity of a transmission, message, or originator. Nonrepudiation concerns proof of delivery and the sender’s identity to establish verification of completion of communication.

On the other hand, Hafiz et al. [6] criticize the cube model: “The safeguards viewpoint of the McCumber cube is effective if the model is used for assessment and management of risks, but it does not provide an effective partition for the classification of security patterns.”

While the cube model and its extension specify the required security-related controls, they do not give a clue about how, what, and where to perform these controls. Assuming a security auditing, how is information collected? Is it done all at once for the whole information system or collected in, say, hierarchical fashion with each subsystem targeted independently? Where to set up the monitors? Should they be located according to functionalities, interfaces, procedures, etc.? Our approach is to bind the types of security-related controls, such as the 27 information types in the cube model with the circulation system of information that we call “information stream.” We use the term “information stream” instead of “information flow” to avoid confusion with models based on information flow models of security. It denotes the intuitive meaning of flowing and streaming of information, rather than the information theory style of information flow. Information stream is the flow of information throughout different stages of an information system. In such a system, information is created, gathered, processed, disclosed, and communicated; thus, control is setup at appropriate locations in these stages.

Before introducing our model, we modify the extended model in two ways:
(1) We extend the information states dimension to five main stages: information creation, collection, processing, disclosing, and communicating (transmitting), where each may have substages (e.g., store, use).
(2) We analyze the extended cube model, and propose that authentication and nonrepudiation represent aspects of transmitting and, hence, belonging to the information states dimension.

Our extended model is based on an information stream that provides fine-grained subprocesses of information handling, it is thus an opportunity to tailor security aspects according to the requirements of each subprocess.

III. INFORMATION STATES

It is important to understand what an “information state” is. Such a notion is the base of one dimension in the cube model. We claim that what is called information state is in fact a stage in information stream caused by “acts on” information.

States in the cube model are not states

The cubic model declares three information states: transmission, storage, and processing. What is the “information state”? According to the originator of the cube model,

As the compound H2O can be water, ice or steam, information has three basic states. At any given moment, information is being transmitted, stored, or processed. The three states exist irrespective of the media in which information resides.[9]. Is this analogy between states of water and states of information valid? The states of H2O have “uniform chemical composition and physical properties.” Being “ice,” “liquid,” and “vapor” are features of H2O. However, being “stored,” “processed,” and “transmitted” are not features of information. When you observe H2O you can decide whether it is in the ice, liquid, or vapor states. When you examine information, you cannot decide whether it is stored, processed, or transmitted; this can be determined only if you examine its “context.”

Many things can be stored, processed and transmitted: water, rocks, sand, grapes, tales, songs, etc. These types of processes reflect an actors-context where actors take the roles of agents and patients. In the information-centric framework, such as the cube model, information plays the roles of patient and usee. Figure 3 includes an agent who acts on information as a patient, including the acts of storing information, processing information, and transmitting information.
The cube model does not distinguish between *acting* on information and *using* information as in using information in decision-making. Decision-making is *use* of information, for example, a security office may analyze (act on) information to help make a decision to abandon certain security services. The agent *uses* information to “abandon certain security services,” as shown in figure 4.

*Use* involves a third actor the *usee*, in addition to the agent and patient, as shown in figure 4. The *usee* is the entity used by the user to act on a patient. For example, a physician uses information to treat a sick person. Storing, processing, and transmitting are really *acts* on information that can be viewed as stages in the information

**Are Authentication and Nonrepudiation states?**

The extended cube model locates authentication and nonrepudiation in terms of confidentiality, integrity, and availability. But authentication of information is an *act* on information. Authenticating information is determining whether it is, in fact, what it is declared to be. It is not a characteristic of information; rather, it involves an actor determining whether it (the information: the patient in the agent/patient model) is not fake (e.g., origin, content, etc.). Similarly, nonrepudiation is an act to document, in a certain way, the transmission of information.

Accordingly, we propose a new model based on information stream that reconstructs the “states” dimension in the original cube model according to the above discussion. The information stream model was first introduced by Al-Fedaghi [1, 2] in the context of privacy. We have modified Al-Fedaghi’s information model to be applicable to a security context and to include communication aspects of the information stream. We then employ the resultant information stream model to construct the “states” dimension in the cube model.

**IV. INFORMATION STREAM MODEL (ISM)**

Information stream is the flow of information throughout different stages of an information system. An information system is a generic term that refers to the circulation system of information analogous to the model of circulation of water among its various compartments in the environment. In such a system, information is created, gathered, processed, disclosed, and communicated. It can dissipate inside any of these stages. These main stages may include secondary stages such as storing and using. A computer-based information system is an information stream system that includes collecting (input), creating (e.g., through data mining), processing, disclosing, and transmitting information.

Identifying the information stream is necessary for recognizing important features such as sources (and sinks), types of information stages and chains of events involving information. The effects of a certain rule might manifest themselves differently at different stages within an information stream. By identifying valid sequences of stages we can impose appropriate constraints at the entry and exit of each stage of the information stream. An information stream model would be a useful tool. Consider that you have been assigned the job of building a system for analyzing a customer’s privacy portfolio based on his or her personal information. You must determine the flow of personal information in a factory-like fashion starting from who is going to be supplying information (the source, e.g., the customer, an agent, a mining program), whether the information is to be stored, processed, disclosed, transmitted, etc., ending with the information exit point in the flow system. Our information stream model (ISM) divides functionality of handling information into stages that include informational entities and processes, as shown in figure 5.
Information in the form of data moves between stages through pipelines denoted by letters in the figure indicating the flow and direction of information from one stage to another. An information agent is represented by a region that includes a complete ISM.

**Example:** Figure 6 shows three regions of three agents. Information flows from agent 1 (e.g., applying for credit on purchase) to agent 2 (e.g., store) to agent 3 (e.g., credit issuer). For example, agent 3 informs agent 2 of the refusal to extend credit because he/she (agent 1) is a risk (generated information). Agent 2 communicates this to the applicant (agent 1). Irrelevant details and unused acts are deleted in figure 6.

Next, we describe different stages, in general. New information is *created* by an agent (e.g., medical diagnostics by physicians, a customer completing a form) or by a machine or deduced by someone (e.g., data mining that generates new information from existing information). The created information enters (L) a use stage (e.g., decision making) or a store stage (K), or it is immediately disclosed (C), which implies an exit point in the information stream system. For example, information moves in a hospital information system to be *used* by a physician in treating a patient. Treating a patient is a use of information, not an act on information. So Use, in ISM, may include many uses such as delivery, treatment, purchasing, etc.

**Creating Stage**

Created information is produced by the agent itself (N) (reporting, as in the case of a newspaper reporter) or automatically (e.g., mining program used by the agent) (M).

The creating stage is the source where new information enters the infosphere of the agent. Three types of possible sources can produce new information:

- **External sources:** Other agents (B)
- **Interior sources:** Interior to the agents manually (N) or automatically (K).

An agent may be an enterprise that includes many people. Any of these workers may create new information if he/she is not designated as an information agent. However, information agents may form a hierarchy of agents where some are sub-agents of higher level agents, as in the case of the departments of an enterprise. The sub-agent may be a single individual or a sub-enterprise. For example, a manager of a department can act as the agent that represents a department in a company and creates information, as in figure 7.

**Processing Stage**

The processing stage involves acting on (e.g., anonymization, data mining, summarizing, translating) information. Processing of information is performed on acquired information from collected information (I) or by creating information (P). Acquisition of information is the internal handling of information where information flows from the collecting/creating stages to the processing stage. The processing stage is where information is modified in form or content. Two types of mining are shown in figure 5, “implied information” mining (back arrow J) and “new information” mining (M). The first type of mining generates implied information (e.g., transitivity). The second type of mining generates new information, as in using categorization of other persons’ information to generate that John is a risk. Processing also includes other types of processing that do not generate implied or new information, but only change the appearance of information as comparing, compressing, translating, etc.
Disclosure and Collecting Stages

Information moves between agents through a pipeline formed from a pair of stages: coupled disclosing/collecting stages, indicating the flow of information from one agent to another. The collecting stage is the information acquisition stage that accepts information from external suppliers and injects it into the agent’s system. The collecting stage includes the possibility of using the collected (raw) information, thus use, in figure 5, is information exit of the system (e.g., address to guide product delivery). The disclosure stage involves releasing information to other agents.

Transmitting Stage

Transmitting refers to acts related to movement of information through the channels of communication between two information agents: disclosing agent and collecting agent. It is a two-sided boundary, where each agent acts on information: one disclosing and the other collecting. The acts of disclosing, transmitting, and collecting are implicitly tied together in this sequence. A multi-agent communication is reduced in our model to that of binary communication. Two acts on the transmitted information can be identified: authentication and nonrepudiation as described previously.

The sequence of acts on information can be formalized as information chains [4] that can be used in different areas such as privacy access control [2].

V. MODEL USAGE

Maconachy et al. [7] proposed applying their extended cube model to develop secure systems through the understanding of IA components and their interaction, and by determining how to protect information in various states. According to Maconachy et al.,

When an analyst is designing or analyzing a system, this framework insures that he does not neglect the interplay of security services, countermeasures, states and time. Research has shown that many countermeasures are the equivalent to putting a tollbooth the middle of the desert. It is just as easy to go around [as] to try to go through the countermeasure and so does effectively increase the system security [7].

Our conceptualization of information stream covers the totality of information stream in an information-shared environment. The ISM model closely complements the mental models that people have of protection, though, by looking after information as it propagates from its origin upon entering the system until it leaves it.

ISM can be used to identify the sources of information security problems. The effects of an information security problem might appear at different stages. The inter-relationships among different manifestations of the security problem can be determined in order to build an airtight security system of information stream that includes all relevant points of possible breaches of security. Leakage of information can be corrected at the creating, storage, usage, processing, disclosing, and communicating stages. Security measures can be applied across all stages and applied to all levels: technological, policy, and human. Security requirements and policy validations can be imposed at the entry and exit to/from each stage, thus, it is possible to trace the information flow across stages to identify the location of the security problem.

In the next section, we utilize these and other features of the ISM model to assign positions and duties to security agents throughout the information stream. A security agent is a reference monitor that enforces the system's security policy, such as by checking proper authorization before granting access to its own domain. A reference monitor is a well known security notion hence, our contribution here is to identify the locations of sensitive security points (checkpoints) throughout the flow of information.

VI. ISM-BASED ASSURANCE MODEL

Our ISM-based model includes three dimensions:

1. Information Characteristics, with three aspects: Confidentiality, Integrity, and Availability.
2. Security Measures, including technology, policies, and procedures, and human factors.
3. Information stream stages comprising 14 stages as shown in table 1.

The ISM-based model includes 3 times 3 times 14 cells, thus represent 126 possible cells.

<table>
<thead>
<tr>
<th>TABLE I</th>
</tr>
</thead>
<tbody>
<tr>
<td>The 14 stages and sub-stages in ISM.</td>
</tr>
<tr>
<td>Creating</td>
</tr>
<tr>
<td>Store</td>
</tr>
<tr>
<td>Use</td>
</tr>
<tr>
<td>Processing</td>
</tr>
<tr>
<td>Store</td>
</tr>
<tr>
<td>Use</td>
</tr>
<tr>
<td>Mining</td>
</tr>
<tr>
<td>Collecting</td>
</tr>
<tr>
<td>Store</td>
</tr>
<tr>
<td>Use</td>
</tr>
<tr>
<td>Disclosing</td>
</tr>
<tr>
<td>Communicating</td>
</tr>
<tr>
<td>Authentication</td>
</tr>
<tr>
<td>Nonrepudiation</td>
</tr>
</tbody>
</table>
The main Information stream stages are the creating, processing, collecting, disclosing, and communicating stages. The store and use sub-stages appear in the collecting, processing, and creating stages. This indicates that collected (raw) data, processed data, and created data have independent storage systems. Similarly, uses in reference to these three types of information are independent from each other. The blank entry represents aspects of the main stage that are not related to store and use.

VII. ISM-BASED SECURITY

In the information world, it is not true that “the best defense is a good offense.” Rather, the best defense is a good defense. A good defense is composed of well-coordinated mechanisms that provide built-in structural multilevel barriers to protect the integrity, confidentiality, and continuous availability of information in the information system. When a barrier has been broken, regardless of the nature of the assault, security features rapidly reestablish security zones through restoration of informational stages as information propagates through the information system.

Protection measures concern outsiders and insiders. We next we identify the points of “check points” in the information stream according to these two types of measures.

Figure 8 shows a network of information stream in which vertices represent the 14 stages and sub-stages given in table 1 and an undirected edge represents an act of information that derives information between two stages. The edges will be called passages.

The directed edges indicate points of information entry and exit in the system as follows:
Points 1, 2 and 6: End points for information stream where information is used.
Point 3: Information is created using non-mining processes such as manual or machine generated weather data.
Point 4: Information enters from another agent and goes to the collecting stage.
Point 5: Information exported to another agent.

Our security architecture requires assigning “security agents” to stages and passages. Thus the required security agents comprise 14 for stages and 17 for passages. These agents may act independently or in concert with other agents. A passage security agent is a security utility that prevents security breaches by protecting “endpoint systems”: the two connected stages as shown in figure 9. This includes policies, monitoring and reporting, filtering, detecting and blocking, analysis features, etc. A stage security agent prevents security breaches by protecting the stage information system itself.

Protection measures concern outsiders and insiders. Next, we identify the points of “check points” in the information stream according to these two types of measures.

Foreign and Domestic Information

As in classical security systems, a basic differentiating between the subjects of security involves classifying them according to their origins: foreign and domestic. “Foreigners” are checked not only at a border point, but also at critical points, such as when changing their places or statuses. Similarly, ISM provides a systematic way of tracking imported information.
Foreign Information

Figure 10 shows a general view of an information system, denoted as S, and points of information entry/exit into/out of the system. Unnecessary detail in the ISM model is not shown.

According to the figure, two types of boundary areas have to be secured:
(1) The boundary point of importing information into S is X1 (blank circles).
(2) The boundary points of exporting information from S are Y1, Y2, Y3, and Y4 (black circles).

The information importing point, X1, injects information in the creation stage. It denotes that an “outside agent” has entered information into the system. The creating and collecting stages form “fronts” for defense against two types of vulnerability:
(a) Using/storing foreign data: Raw data may be corrupted, illegally acquired, illegally used, etc.
(b) Penetrating foreign data into the processing module of the system: Processing incorrect data, bombs, etc.

Figure 11 shows points where “check points” or security agents can be installed. The model assumes that collected data, created data, and processed data each has its own storage system, without mixing these types of data. Thus, the mere act of storing bad raw data (P6 inside the store boxes) is a security risk that must be watched out for. In addition, risk may involve the mere use of collected information without reaching the processing stage.

Examples of specific security measures used by different security agents are as follows:
P1: System authorizations (e.g., wrong destination) as the receiving party of data, recovering from transmission failures.
P2: Anti-virus, hacking watch, firewalls, recovery, monitoring.
P5: Access control
P6: Preventing accidental loss of data, updating and maintenance, encryption, backups.

Domestic Information

Turning to information generated inside the system, figure 12 shows a general view of points of information entry and exit. Information is interiorly created at the creating stage and consumed in the processing and disclosing stages. The collecting stage is omitted because the system S cannot be a collection agent of information generated by S itself. For new information, we cannot say that S collects what it generates. By default, the generated information is in the possession of S.

At X1, information is generated by the information agent (e.g., reception employee). At X2, information is generated by the information agent automatically.

Figure 13 shows points where security agents can be installed to protect against internally generated vulnerability. At the creating stage, a security agent is necessary at the storage place of the created data. This guard checks the integrity of the created data.
VIII. CONCLUSION

One of the important aspects of security organization is to establish a framework to identify significant security points where policies and procedures are declared. The (information) security infrastructure comprises entities, processes, and technology. All are participants in handling information, which is the item that needs to be protected. Our approach is to identify information stream as the principal focus of security consideration. It is based on identifying points of transformation in the flow of information as the pivots around which security organization is built.

Currently, there is an implicit understanding of such a method. For example, in a computer system critical security posts are built around a core set of functionalities such as access control of stored information, internal processes, logon accounts, external communication connections, etc. Inside a single system (e.g., database security) security functions are unsystematically allocated according to information type, operation type, output type (e.g., mining), etc. The information stream model introduces a general coherent blueprint of security-significant posts that complements identification of security-related controls.

REFERENCES

Extended Entity-Relationship Model

Bernhard Thalheim
Christian-Albrechts University Kiel, http://www.informatik.uni-kiel.de/~thalheim

SYNONYMS
EERM, HERM; higher-order entity-relationship model; hierarchical entity-relationship model

DEFINITION
The extended entity-relationship (EER) model is a language for defining the structure (and functionality) of database or information systems. Its structure is developed inductively. Basic attributes are assigned to base data types. Complex attributes can be constructed by applying constructors such as tuple, list or set constructors to attributes that have already been constructed. Entity types conceptualise structuring of things of reality through attributes. Cluster types generalise types or combine types into singleton types. Relationship types associate types that have already been constructed into an association type. The types may be restricted by integrity constraints and by specification of identification of objects defined for a type. Typical integrity constraints of the extended entity-relationship model are participation, look-across, and general cardinality constraints. Entity, cluster, and relationship classes contain a finite set of objects defined on these types. The types of an EER schema are typically depicted by an EER diagram.

HISTORICAL BACKGROUND
The entity-relationship (ER) model was introduced by P.P. Chen in 1976 [1]. The model conceptualises and graphically represents the structure of the relational model. It is currently used as the main conceptual model for database and information system development. Due to its extensive usage a large number of extensions to this model were proposed in the 80’s and 90’s. Cardinality constraints [1, 3, 4, 8] are the most important generalisation of relational database constraints [7]. These proposals have been evaluated, integrated or explicitly discarded in an intensive research discussion. The semantic foundations proposed in [2, 5, 8] and the various generalisations and extensions of the entity-relationship model have led to the introduction of the higher-order or hierarchical entity-relationship model [8] which integrates most of the extensions and also supports conceptualisation of functionality, distribution [9], and interactivity [6] for information systems. Class diagrams of the UML standard are a special variant of extended entity-relationship models.
The ER conferences (annually; since 1996: International Conference on Conceptual Modeling, http://www.conceptualmodeling.org/) are the main forum for conceptual models and modelling.

SCIENTIFIC FUNDAMENTALS
The extended entity-relationship model is mainly used as a language for conceptualisation of the structure of information systems applications. Conceptualisation of database or information systems aims to represent the logical and physical structure of an information system. It should contain all the information required by the user and required for the efficient behavior of the whole information system for all users. Conceptualisation may further target the specification of database application processes and the user interaction. Structure description are currently the main use of the extended ER model.

An example of an EER diagram.
The EER model uses a formal language for schema definition and diagrams for graphical representation of the
schema. Let us consider a small university application for management of Courses. Proposed courses are based on courses and taught by a docent or an external docent within a certain semester and for a set of programs. Proposals typically include a request for a room and for a time and a categorisation of the kind of the course. Theses proposals are the basis for course planning. Planning may change time, room and kind. Planned courses are held at the university. Rooms may be changed. The example is represented by the EER diagram in Figure 1.

![Figure 1: Extended Entity-Relationship Diagram for Course Management](image)

Entity types are represented graphically by rectangles. Attribute types are associated with the corresponding entity or relationship type. Attributes primarily identifying a type are underlined. Relationship types are represented graphically by diamonds and associated by directed arcs to their components. A cluster type is represented by a diamond, is labelled by the disjoint union sign, and has directed arcs from the diamond to its component types. Alternatively, the disjoint union representation ⊕ is attached to the relationship type that uses the cluster type. In this case directed arcs associate the ⊕ sign with component types. An arc may be annotated with a label.

### The definition scheme for structures.

The extended entity-relationship model uses a data type system for its attribute types. It allows the construction of entity types \( E \equiv (\text{attr}(E), \Sigma_E) \) where \( E \) is the entity type defined as a pair — the set \( \text{attr}(E) \) of attribute types and the set \( \Sigma_E \) of integrity constraints that apply to \( E \). The definition \( def \) of a type \( T \) is denoted by \( T \equiv def \).

The EER model lets users inductively build relationship types \( R \equiv (T_1, \ldots, T_n, \text{attr}(R), \Sigma_R) \) of order \( i \ (i \geq 1) \) through a set of (labelled) types of order less than \( i \), a set of attribute types, and a set of integrity constraints that apply to \( R \). The types \( T_1, \ldots, T_n \) are the components of the relationship type. Entity types are of order 0. Relationship types are of order 1 if they have only entity types as component types. Relationship types are of order \( i \) if all component types are of order less than \( i \) and if one of the component types is of order \( i - 1 \).

Additionally, cluster types \( C \equiv T_{1} \cup \ldots \cup T_{n} \) of order \( i \) can be defined through a disjoint union \( \cup \) of relationship types of order less than \( i \) or of entity types.

Entity/relationship/cluster classes \( T^C \) contain a set of objects of the entity/relationship/cluster type \( T \). The EER model mainly uses set semantics, but (multi-)list or multiset semantics can also be used. Integrity constraints apply to their type and restrict the classes. Only those classes are considered for which the constraints of their types are valid. The notions of a class and of a type are distinguished. Types describe the structure and constraints. Classes contain objects.

The data type system is typically inductively constructed on a base type \( B \) by application of constructors such as the tuple or products constructor \((\ldots)\), set constructor \(\{\ldots\}\), and the list constructor \(<\ldots>\). Types may be optional component types and are denoted by \([\ldots]\).

The types \( T \) can be labelled \( l \cdot T \). The label is used as an alias name for the type. Labels denote roles of the type. Labels must be used if the same type is used several times as a component type in the definition of a relationship.
or cluster type. In this case they must be unique.

An entity-relationship schema consists of a set of data, attribute, entity, relationship, and cluster types which types are inductively built on the basis of the base types. Given a base type system \( B \). The types of the ER schema are defined through the type equation:

\[
T = B \mid (l_1:T,\ldots,l_n:T) \mid \{T\} \mid \langle T \rangle \mid [T] \mid T \cup T \mid l:T \mid N \models T
\]

Structures in detail.
The classical four-layered approach is used for inductive specification of database structures. The first layer is the data environment, called the basic data type scheme, which is defined by the system or is the assumed set of available basic data types. The second layer is the schema of a database. The third layer consists of the macro-data that are generated from the micro-data by application of view queries to the micro-data.

Attribute types and attribute values.
The classical ER model uses basic (first normal form) attributes. Complex attributes are inductively constructed by application of type constructors such as the tuple constructor \( (..) \), set constructor \( \{..\} \), and the list constructor \( <..> \). Typical base types are integers, real numbers, strings, and time. Given a set of names \( \mathcal{N} \) and a set of base types \( B \), a basic attribute type \( A :: B \) is given by an (attribute) name \( A \in \mathcal{N} \) and a base type \( B \). The association between the attribute name and the underlying type is denoted by \( :: \). The base type \( B \) is often called the domain of \( A \), i.e. \( dom(A) = B \). Complex attributes are constructed on base attributes by application of the type constructors. The notion of a domain is extended to complex attributes, i.e. the domain of the complex attribute \( A \) is given by \( dom(A) \). Components of complex attributes may be optional, e.g., the \( Title \) in the attribute \( Name \).

Typical examples of complex and basic attributes in Figure 1 are

- \( Name \equiv (\text{FirstNames}, \text{FirstName}) \), \( FamName \), \( \text{[AcadTitles]} \), \( \text{[FamilyTitle]} \)
- \( PersNo \equiv \text{EmplNo} \cup \text{SocSecNo} \)
- \( AcadTitles \equiv \{\text{AcadTitle}\} \)
- \( Contact \equiv (\text{Phone}(\{\text{PhoneAtWork}\}), \text{private}), \text{Email}, \text{URL}, \text{WebContact}, [\text{Fax}(\{\text{PhoneAtWork}\})] ) \)
- \( PostalAddress \equiv (\text{Zip}, \text{City}, \text{Street}, \text{HouseNumber}) \)
- \( 	ext{DateOfBirth} :: \text{date}, \text{AcadTitle} :: \text{acadTitleType}, \text{FamilyTitle} :: \text{familyTitleAcronym} \)
- \( \text{Zip} :: \text{string7}, \text{SocSecNo} :: \text{string9}, \text{EmplNo} :: \text{int}, \text{City} :: \text{varString}, \text{Street} :: \text{varString}, \text{HouseNumber} :: \text{smallInt} \)

The complex attribute \( Name \) is structured into a sequence of first names, a family name, an optional complex set-valued attribute for academic titles, and an optional basic attribute for family titles. Academic titles and family titles can be distinguished from each other.

Entity types and entity classes.

Entity types are characterized by their attributes and their integrity constraints. Entity types have a subset \( K \) of the set of attributes which serve to identify the objects of the class of the type. This concept is similar to the concept of key known for relational databases. The key is denoted by \( ID(K) \). The set of integrity constraints \( \Sigma_E \) consists of the keys and other integrity constraints. Identifying attributes may be underlined instead of having explicit specification.

Formally, an entity type is given by a name \( E \), a set of attributes \( \text{attr}(E) \), a subset \( id(E) \) of \( \text{attr}(E) \), and a set \( \Sigma_E \) of integrity constraints, i.e.

\[
E \equiv (\text{attr}(E), \Sigma_E)
\]

The following types are examples of entity types in Figure 1:

- \( Person \equiv (\{ Name, Login, URL, Address, Contact, DateOfBirth, PersNo \}) \)
- \( Course \equiv (\{ CourseID, Title, URL \}, \{ ID(\{CourseID\})\}) \)
- \( Room \equiv (\{Building, Number, Capacity\}, \{ ID(\{Building, Number\})\}) \)
- \( Semester \equiv (\{Term, Date(Starts, Ends)\}, \{ ID(\{Term\})\}) \)

An ER schema may use the same attribute name with different entity types. For instance, the attribute \( URL \) in Figure 1 is used for characterising additional information for the type \( Person \) and the type \( Course \). If they need to be distinguished, then complex names such as \( CourseURL \) and \( PersonURL \) are used.

Objects on type \( E \) are tuples with the components specified by a type. For instance, the object (or entity) \((HRS3, 408A, 15)\) represents data for the \( Room \) entity type in Figure 1.
An entity class $E^C$ of type $E$ consists of a finite set of objects on type $E$ for which the set $\Sigma_E$ of integrity constraints is valid.

**Cluster types and cluster classes.**

A disjoint union $\cup$ of types whose identification type is domain compatible is called a cluster. Types are domain compatible if they are subtypes of a common more general type. The union operation is restricted to disjoint unions since identification must be preserved. Otherwise, objects in a cluster class cannot be related to the component classes of the cluster type. Cluster types can be considered as a generalisation of their component types.

A cluster type (or “category”)

$$C \doteq l_1 : R_1 \cup l_2 : R_2 \cup \ldots \cup l_k : R_k$$

is the (labelled) disjoint union of types $R_1, \ldots, R_k$. Labels can be omitted if the types can be distinguished.

The following type is an example of a cluster type:

$$Teacher \doteq ExternalDocent : CollaborationPartner \cup Docent : Professor.$$

The cluster class $C^C$ is the ‘disjoint’ union of the sets $R_1^C, \ldots, R_k^C$. It is defined if $R_1^C, \ldots, R_k^C$ are disjoint on their identification components. If the sets $R_1^C, \ldots, R_k^C$ are not disjoint then labels are used for differentiating the objects of clusters. In this case, an object uses a pair representation $(i, o_i)$ for objects $o_i$ from $R_i^C$.

**Relationship types and relationship classes.**

First order relationship types are defined as associations between entity types or clusters of entity types. Relationship types can also be defined on the basis of relationship types that are already defined. This construction must be inductive and cannot be cyclic. Therefore, an order is introduced for relationship types. Types can only be defined on the basis of types which have a lower order.

For instance, the type $Professor$ in Figure 1 is of order 1. The type $ProposedCourse$ is of order 2 since all its component types are either entity types or types of order 1. A relationship type of order $i$ is defined as an association of relationship types of order less than $i$ or of entity types. It is additionally required that at least one of the component types is of order $i - 1$ if $i > 1$. Relationship types can also be characterized by attributes. Relationship types with one component type express a subtype or an Is-A relationship type. For instance, the type $Professor$ is a subtype of the type $Person$.

Component types of a relationship type may be labelled. Label names typically provide an understanding of the role of a component type in the relationship type. Labelling uses the definition scheme Label : Type. For instance, the $Kind$ entity type is labelled by $Proposal$ for the relationship type $ProposedCourse$ in Figure 1.

Cluster types have the maximal order of their component types. Relationship types also may have cluster type components. The order of cluster type components of a relationship type of order $i$ must be less than $i$.

Component types that are not used for identification within the relationship type can be optional. For instance, the $Room$ component in Figure 1 is optional for the type $PlannedCourse$. If the relationship object in the $PlannedCourse$ class does not have a room then the proposal for rooms in $ProposedCourse$ is accepted. A specific extension for translation of optional components may be used. For instance, $Room$ in Figure 1 is inherited to $PlannedCourse$ from $ProposedCourse$ if the $Room$ component for a $PlannedCourse$ is missing.

Higher order types allow a convenient description of types that are based on other types. For example, consider the course planning application in Figure 1. Lectures are courses given by a professor or a collaboration partner within a semester for a number of programs. Proposed courses extend lectures by describing which room is requested and which time proposals and which restrictions are made. Planning of courses assigns a room to a course that has been proposed and assigns a time frame for scheduling. The kind of the course may be changed. Courses that are held are based on courses planned. The room may be changed for a course. The following types specify these assertions.

$ProposedCourse \doteq \{ (Teacher, Course, Proposal: Kind, Request: Room, Semester, Set2: \{ Program \},
\{ Time(Proposal, SideCondition) \}, \Sigma_{ProposedCourse} \}.$

$PlannedCourse \doteq \{ ProposedCourse, [Reassigned: Kind], [Reassigned: Room],
\{ TimeFrame, TermCourseID \}, \Sigma_{PlannedCourse} \}.$

$CourseHeld \doteq \{ PlannedCourse, \{ Reassigned: Room \}, \{ StartDate, EndDate, AssistedBy \},\Sigma_{CourseHeld} \}.$

The second and third types use optional components in case a proposal or a planning of rooms or kinds is changed. Typically, planned courses are identified by their own term-specific identification. Integrity constraints can be
omitted until they have been defined.

Formally, a relationship type is given by a name \( R \), a set \( \text{compon}(R) \) of labelled components, a set of attributes \( \text{attr}(R) \), and a set \( \Sigma_R \) of integrity constraints that includes the identification of the relationship type by a subset \( \text{id}(R) \) of \( \text{compon}(R) \cup \text{attr}(R) \), i.e.,

\[
R \equiv (\text{compon}(R), \text{attr}(R), \Sigma_R) .
\]

It is often assumed that the identification of relationship types is defined exclusively through their component types. Relationship types that have only one component type are unary types. These relationship types define subtypes. If subtypes need to be explicitly represented then binary relationship types named by \( \text{IsA} \) between the subtype and the supertype are used. For instance, the type \( \text{Professor} \) in Figure 1 is a subtype of the type \( \text{Person} \). An object (or a “relationship”) on the relationship type \( R \equiv (R_1, ..., R_n, \{B_1, ..., B_k\}, \text{id}(R), \Sigma_R) \) is an element of the Cartesian product \( R_1^C \times ... \times R_n^C \times \text{dom}(B_1) \times ... \times \text{dom}(B_k) \). A relationship class \( R^C \) consists of a finite set \( R^C \subseteq R_1^C \times ... \times R_n^C \times \text{dom}(B_1) \times ... \times \text{dom}(B_k) \) of objects on \( R \) for which \( \text{id}(R) \) is a key of \( R^C \) and which obeys the constraints \( \Sigma_R \).

**Integrity constraints.**

Each database model also uses a set of implicit model-inherent integrity constraints. For instance, relationship types are defined over their component types, and a (relationship) object presumes the existence of corresponding component objects. Typically only finite classes are considered. The EER schema is acyclic. Often names or labels are associated with a minimal semantics that can be derived from the meaning of the words used for names or labels. This minimal semantics allows us to derive synonym, homonym, antonym, troponym, hyponym, and holonym associations among the constructs used.

The most important class of integrity constraints of the EER model is the class of cardinality constraints. Other classes of importance for the EER model are multivalued dependencies, inclusion and exclusion constraints and existence dependencies[7]. Functional dependencies, keys and referential constraints (or key-based inclusion dependencies) can be expressed through cardinality constraints.

Three main kinds of cardinality constraints are distinguished: participation constraints, look-across constraints, and general cardinality constraints. Given a relationship type \( R \equiv (\text{compon}(R), \text{attr}(R), \Sigma_R) \), a component \( R' \) of \( R \), the remaining substructure \( R'' = R \setminus R' \) and the remaining substructure \( R''' = R'' \cap_R \text{compon}(R) \) without attributes of \( R \).

The participation constraint \( \text{card}(R, R') = (m,n) \) restricts the number of occurrences of \( R' \) objects in the relationship class \( R^C \) by the lower bound \( m \) and the upper bound \( n \). It holds in a relationship class \( R^C \) if for any object \( o' \in R^C \) there are at least \( m \) and at most \( n \) objects \( o \in R^C \) with \( \pi_{R'}(o) = o' \) for the projection function \( \pi_{R'} \) that projects \( o \) to its \( R' \) components.

Participation constraints relate objects of relationship classes to objects of their component classes. For instance, the constraint \( \text{card}(\text{ProposedCourse}, \text{SemesterCourse}) = (0,3) \) restricts relationship classes for proposals for courses per semester to at least 0 and at most 3, i.e. each course is proposed at least 3 times in a semester. There are at most three objects \( o \) in \( \text{ProposedCourse}^C \) with the same course and semester objects. The integrity constraint \( \text{card}(\text{ProposedCourse}, \text{DocentSemester}) = (3,7) \) requires that each docent is giving at least 3 courses and at most 7 courses. External docents may be obliged by other restrictions, e.g., \( \text{card}(\text{ProposedCourse}, \text{ExternalDocentSemester}) = (0,1) \).

Formally, the integrity constraint \( \text{card}(R, R') = (m,n) \) is valid in \( R^C \) if \( m \leq \frac{|\{o \in R^C : \pi_{R'}(o) = o'\}|}{n} \) for any \( o' \in \pi_{R'}(R^C) \) and the projection \( \pi_{R'}(R^C) \) of \( R^C \) to \( R' \).

If \( \text{card}(R, R') = (0,1) \) then \( R' \) forms an identification or a key of \( R \), i.e. \( \text{ID}(R') \) for \( R \). This identification can also be expressed by a functional dependency \( R : R' \rightarrow R'' \).

The lookup or look-across constraint \( \text{look}(R, R') = m.n \) describes how many objects \( o'' \) from \( R''' \) may potentially ‘see’ an object \( o' \) from \( R^C \). It holds in a relationship class \( R^C \) if for any object \( o'' \in \text{dom}(R'''(o')) \) there are at least \( m \) and at most \( n \) related objects \( o' \) with \( \pi_{R'}(o) = o' \), i.e. \( m \leq \frac{|\{o' \in \pi_{R'}(R^C) : o \in R^C \land \pi_{R'}(o) = o' \land \pi_{R''}(o) = o''\}|}{n} \) for any \( o'' \in \text{dom}(R'''(o')) \). Typically, look-across constraints are used for components consisting of one type. Look-across constraints are not defined for relationship types with one component type.

Look-across constraints are less intuitive for relationship types with more than 2 component types or with attribute types. For instance, the look-across constraint \( \text{look}(\text{ProposedCourse}, \text{DocentSemester}) = 0..7 \) specifies that for any combination of \( \text{Teacher} \), \( \text{Room} \), \( \text{Kind} \), and \( \text{Program} \) objects there are between 0 and 7 \( \text{Docent} \) and \( \text{Semester} \).
relationships. The lower bound expresses that there are Teacher, Room, Kind, and Program which do not have a Docent and Semester combination.

Look-across constraints for a binary relationship type which component types form a key of the relationship type can equivalently expressed by participation constraints, i.e. look$(R, R_1) = m_1 \ldots n_1$ if and only if card$(R, R_2) = (m_1, n_1)$. Similarly, look$(R, R_2) = m_2 \ldots n_2$ if and only if card$(R, R_1) = (m_2, n_2)$. This equivalence is neither valid for binary relationship types which cannot be identified by their components and nor for relationship types with more than 2 components.

Participation and look-across constraints can be extended to substructures and intervals and to other types such as entity and cluster types. Given a relationship type $R$, a substructure $R'$ of $R$, $R''$ and $R'''$ as above. Given furthermore an interval $I \subseteq \mathbb{N}_0$ of natural numbers including 0. The (general) cardinality constraint card$(R, R') = I$ holds in a relationship class $RC$ if for any object $o' \in \pi_R(RC)$ there are $i \in I$ objects $o$ with $\pi_R(o) = o'$, i.e. $| \{ o \in RC : \pi_R(o) = o' \} | \in I$ for any $o' \in \pi_R(RC)$.

The following participation, look-across and general cardinality constraints are examples in Figure 1:

- For any $R' \in \{ \text{Semester, Course, Kind} \} \quad \text{card}(\text{ProposedCourse}, R') = (0, n)$,
- \text{card}(\text{ProposedCourse, Semester Course Teacher}) = (0, 1),
- \text{card}(\text{CourseHeld, PlannedCourse}) = (1, 1),
- \text{card}(\text{PlannedCourse, ProposedCourse[Semester] Room TimeFrame}) = (0, 1),
- \text{card}(\text{ProposedCourse, Docent Semester}) = \{0, 3, 4, 5, 6, 7\}.

The first constraint does not restrict the database. The second constraint expresses a key or functional dependency. The types Semester Course Teacher identify any of the other types in the type ProposedCourse, i.e. 

\text{ProposedCourse : } \{ \text{Semester, Course, Teacher} \} \rightarrow \{ \text{Request, Time, Proposal, Set2} \}.

The third constraint requires that any planned course must be given. The fourth constraint requires that rooms are not overbooked. The fifth constraint allows that docents may not teach in a semester, i.e. have a sabbatical. If a docent is teaching in a semester then at least 3 and at most 7 courses are given by the docent.

Look-across constraints were originally introduced by P.P. Chen [1] as cardinality constraints. UML uses look-across constraints. Participation and look-across constraints cannot be axiomatised through a Hilbert- or Gentzen-type logical calculus. If only upper bounds are of interest then an axiomatisation can be found in [3] and [4]. General cardinality constraints combine equality-generating and object-generating constraints such as keys, functional dependencies and referential integrity constraints into a singleton construct.

Logical operators can be defined for each type. A set of logical formulas using these operators can define the integrity constraints which are valid for each object of the type.

**Schemata.**

The schema is based on a set of base (data) types which are used as value types for attribute types.

A set $\{E_1, \ldots, E_n, C_1, \ldots, C_l, R_1, \ldots, R_m\}$ of entity, cluster and (higher-order) relationship types on a data scheme DD is called schema if the relationship and cluster types use only the types from $\{E_1, \ldots, E_n, C_1, \ldots, C_l, R_1, \ldots, R_m\}$ as components and cluster and relationship types are properly layered. An EER schema is defined by the pair $D = (S, \Sigma)$ where $S$ is a schema and $\Sigma$ is a set of constraints. A database $D^C$ on $D$ consists of classes for each type in $D$ such that the constraints $\Sigma$ are valid.

The classes of the extended ER model have been defined through sets of objects on the types. In addition to sets, lists, multi-sets or other collections of objects may be used. In this case, the definitions used above can easily be extended [8].

A number of domain-specific extensions have been introduced to the ER model. One of the most important is the extension of the base types by spatial data types such as: point, line, oriented line, surface, complex surface, oriented surface, line bunch, and surface bunch. These types are supported by a large variety of functions such as: meets, intersects, overlaps, contains, adjacent, planar operations, and a variety of equality predicates.

The translation of the schema to (object-)relational or XML schemata can be based on a profile [8]. Profiles define which translation choice is preferred over other choices, how hierarchies are treated, which redundancy and null-value support must be provided, which kind of constraint enforcement is preferred, which naming conventions are chosen, which alternative for representation of complex attributes is preferred for which types, and whether weak types can be used. The treatment of optional components is also specified through the translation profile of the types of the schema. A profile may require the introduction of identifier types and base the identification on the identifier. Attribute types may be translated into data formats that are supported by the target system.
The EER schema can be used to define views. The generic functions insert, delete, update, projection, union, join, selection and renaming can be defined in a way similarly to the relational model. Additionally, nesting and unnesting functions are used. These functions form the algebra of functions of the schema and are the basis for defining queries. A singleton view is defined by a query that maps the EER schema to new types. Combined views also may be considered which consist of singleton views which together form another EER schema. A view schema is specified over an EER schema $D$ by a schema $V = \{S_1, \ldots, S_m\}$, an auxiliary schema $A$ and a (complex) query $q : D \times A \to V$ defined on $D$ and $A$. Given a database $D^C$ and the auxiliary database $A^C$. The view is defined by $q(D^C \times A^C)$.

**Graphical representation.**
The schema in Figure 1 consists of entity, cluster and relationship types. The style of drawing diagrams is one of many variants that have been considered in the literature. The main difference of representation is the style of drawing unary types. Unary relationship types are often represented by rectangles with rounded corners or by (directed) binary IsA-relationship types which associate by arcs the supertype with the subtype. Tools often do not allow cluster types and relationship types of order higher than 1. In this case, those types can be objectified, i.e. represented by a new (abstract) entity type that is associated through binary relationship types to the components of the original type. In this case, identification of objects of the new type is either inherited from the component types or is provided through a new (surrogate) attribute. The first option results in the introduction of so-called weak types. The direct translation of these weak types to object-relational models must be combined with the introduction of rather complex constraint sets. Typically, this complexity can be avoided if the abstract entity type is mapped together with the new relationship types to a singleton object-relational type. This singleton type is also the result of a direct mapping of the original higher-order relationship type. The diagram can be enhanced by an explicit representation of cardinality and other constraints. If participation constraints $\text{card}(R, R') = (m, n)$ are used for component consisting of one type $R'$ then the arc from $R$ to $R'$ is labelled by $(m, n)$. If look-across constraints $\text{look}(R, R') = m..n$ are used for binary relationship types then the arc from $R$ to $R'$ is labelled by $m..n$.

**KEY APPLICATIONS**
The main application area for extended ER models is the conceptualisation of database applications. Database schemata can be translated to relational, XML or other schemata based on transformation profiles that incorporate properties of the target systems.

**FUTURE DIRECTIONS**
The ER model has had a deep impact on the development of diagramming techniques in the past and is still influencing extensions of the unified modelling language UML. UML started with binary relationship types with look-across constraints and without relationship type attributes. Class diagrams currently allow n-ary relationship types with attributes. Relationship types may be layered. Cluster types and unary relationship types allow for distinguishing generalisation from specialisation. ER models are not supported by native database management systems and are mainly used for modelling of applications at the conceptual or requirements level. ER schemata are translated to logical models such as XML schemata or relational schemata or object-relational schemata. Some of the specifics of the target models are not well supported by ER models and must be added after translating ER schemata to target schemata, e.g., specific type semantics such as list semantics (XML) or as special ordering or aggregation treatment of online analytical processing (OLAP) applications. The ER model has attracted a lot of research over the last 30 years. Due to novel applications and to evolution of technology old problems and novel problems are challenging the research on this model. Typical old problems that are still not solved in a satisfactory manner are: development of a science of modelling, quality of ER schemata, consistent refinement of schemata, complex constraints, normalisation of ER schemata, normalisation of schemata in the presence of incomplete constraint sets. Novel topics for ER research are for instance: evolving schema architectures, collaboration of databases based on collaboration schemata, layered information systems.
and their structuring, schemata with redundant types, ER schemata for OLAP applications.

Structures of database applications are often represented through ER models. Due to the complexity of applications, a large number of extensions have recently been proposed, e.g., temporal data types, spatial data types, OLAP types and stream types. Additionally, database applications must be integrated and cooperate in a consistent form. The harmonisation of extensions and the integration of schemata is therefore a never ending task for database research.

ER models are currently extended for support of (web) content management that is based on structuring of data, on aggregation of data, on extending data by concepts and on annotating data sets for simple reference and usage. These applications require novel modelling facilities and separation of syntactical, semantical and pragmatic issues. The ER model can be extended to cope with these applications.

The ER model is mainly used for conceptual specification of database structuring. It can be enhanced by operations and a query algebra. Operations and the queries can also be displayed in a graphical form, e.g. on the basis of VisualSQL. Most tools supporting ER models do not currently use this option. Enhancement of ER models by functionality is necessary if the conceptualisation is used for database development. Based on functionality enhancement, view management facilities can easily be incorporated into these tools.

ER models are becoming a basis for workflow systems data. The standards that have been developed for the specification of workflows have not yet been integrated into sophisticated data and application management tools.

URL TO CODE
http://www.informatik.uni-kiel.de/~thalheim/HERM.htm
http://www.is.informatik.uni-kiel.de/~thalheim/indeeerm.htm

Readings on the RADD project (Rapid Application and Database Development)

CROSS REFERENCE
I. DATABASE FUNDAMENTALS
   a. Data models (including semantic data models)
   b. Entity-Relationship (ER) model
   c. Unified modelling language (UML)

III. THEORETICAL ASPECTS
   b. Relational Theory

RECOMMENDED READING
Between 3 and 15 citations to important literature, e.g., in journals, conference proceedings, and websites.

Specialisation and Generalisation

Bernhard Thalheim

Christian-Albrechts University Kiel, http://www.informatik.uni-kiel.de/~thalheim/HERM.htm

SYNONYMS
refinement, abstraction, hierarchies;
clustering, grouping, inheritance

DEFINITION
Generalisation and specialisation are main principles of database modelling. Generalisation maps or groups types or classes to more abstract or combined ones. It is used to combine common features, attributes, or methods. Specialisation is based on a refinement of types or classes to more specific ones. It allows developers to avoid null values and to hide details from non-authorised users. Typically, generalisations and specialisations form a hierarchy of types and classes. The more specialised classes may inherit attributes and methods from more general ones. In database modelling and implementation clusters of types to a type that represents common properties and abstractions from a type are the main kinds of generalisations. Is-A associations that specialise a type to a more specific one and Is-A-Role-Of associations that considers a specific behaviour of objects are the main kinds of specialisations.

MAIN TEXT
Specialisation introduces a new entity type by adding specific properties belonging to that type which are different from the general properties of its more general type. Generalisation introduces the Role-Of relationship or the Is-A relationship between a subtype and its general type. Therefore, the application, implementation, and processes are different. For generalisation the general type must be the union of its subtypes. The subtypes can be virtually clustered by the general type. This tends not to be the case for specialisation. Specialisation is a refinement or restriction of a type to more special ones. Typical specialisations are Is-A and Has-Role associations. Exceptions can be modelled by specialisations.

Different kinds of specialisation may be distinguished: structural specialisation which extends the structure, semantic specialisation which strengthens type restrictions, pragmatical specialisation which allows to separate the different usage of objects in contexts, operational specialisation which introduces additional operations, and hybrid specialisations. Is-A specialisation requires structural and strong semantic specialisation. Is-A-Role-Of specialisation requires structural, pragmatical and strong semantic specialisation.

Generalisation is based either on abstraction or on grouping. The cluster construct of the extended ER model is used to represent generalisations. Generalisation tends to be an abstraction in which a more general type is defined by extracting common properties of one or more types while suppressing the differences between them. These types are subtypes of the generic type. New types are created by generalizing classes that already exist. Structural combination typically assumes the existence of a unifiable identification of all types. Semantical combination allows the disjunction of types through the linear sum of semantics. Pragmatical generalisation is based on building collections whenever applications require a consideration of commonalties.

CROSS REFERENCE
I. DATABASE FUNDAMENTALS
a. Data models (including semantic data models)

REFERENCES
Abstraction

Bernhard Thalheim

Christian-Albrechts University Kiel, http://www.informatik.uni-kiel.de/~thalheim/HERM.htm

SYNONYMS
component abstraction, localisation abstraction, implementation abstraction; association, aggregation, composition, grouping, specialisation, generalisation, classification

DEFINITION
Abstraction allows developers to concentrate on the essential, relevant or important parts of an application. It uses a mapping to a model from things in reality or from virtual things. The model has the truncation property, i.e. it lacks some of the details in the original, and a pragmatic property, i.e. the model use is only justified for particular model users, tools of investigation, and periods of time. Database engineering uses construction abstraction, context abstraction and refinement abstraction. Construction abstraction is based on the principles of hierarchical structuring, constructor composition, and generalisation. Construction abstraction assumes that the surroundings of a concept are commonly assumed by a community or within a culture and focuses on the concept, turning away attention from its surroundings such as the environment and setting. Refinement abstraction uses the principle of modularisation and information hiding. Developers typically use conceptual models or languages for representing and conceptualising abstractions. The enhanced entity-relationship model schema are typically depicted by an EER diagram.

MAIN TEXT
Database engineering distinguishes three kinds of abstraction: construction abstraction, context abstraction and refinement abstraction.
Constructor composition depends on the constructors as originally introduced by J. M. Smith and D.C.W. Smith. Composition constructors must be well founded and their semantics must be derivable by inductive construction. There are three main methods for construction: development of ordered structures on the basis of hierarchies, construction by combination or association, and construction by classification into groups or collections. The set constructors $\subset$ (subset), $\times$ (product) and $\mathcal{P}$ (powerset) for subset, product and nesting are complete for the construction of sets.

Subset constructors support hierarchies of object sets in which one set of objects is a subset of some other set of objects. Subset hierarchies are usually a rooted tree. Product constructors support associations between object sets. The schema is decomposed into object sets related to each other by association or relationship types. Power set constructors support a classification of object sets into clusters or groups of sets - typically according to their properties.

Context abstraction allows developers to commonly concentrate on those parts of an application that are essential for some viewpoints during development and deployment of systems. Typical kinds of context abstraction are component abstraction, separation of concern, interaction abstraction, summarisation, scoping, and focusing on typical application cases.
Component abstraction factors out repeating, shared or local patterns of components or functions from individual concepts. It allows developers to concentrate on structural or behavioral aspects of similar elements of components. Separation of concern allows developers to concentrate on those concepts that are a matter of development and to neglect all other concepts that are stable or not under consideration. Interaction abstraction allows developers to concentrate on those parts of the model that are essential for interaction with other systems or users. Summarisation maps the conceptualisations within the scope to more abstract concepts. Scoping is typically used to select those concepts that are necessary for current development and removes those concepts that do not have an impact on the necessary concepts.

Database models may cover a large variety of different application cases. Some of them reflect exceptional,
abnormal, infrequent and untypical application situations. Focusing on typical application cases explicitly separates models for the normal or typical application case from those that are atypical. Atypical application cases are not neglected but can be folded into the model whenever atypical situations are considered.

The context abstraction concept is the main concept behind federated databases. Context of databases can be characterized by schemata, version, time, and security requirements. Sub-schemata, types of the schemata or views on the schemata, are associated by explicit import/export bindings based on a name space. Parametrisation lets developers to consider collections of objects. Objects are identifiable under certain assumptions and completely identifiable after instantiation of all parameters.

Interaction abstraction allows developers to display the same set of objects in different forms. The view concept supports this visibility concept. Data is abstracted and displayed in various levels of granularity. Summarisation abstraction allows developers to abstract from details that are irrelevant at a certain step. Scope abstraction allows developers to concentrate on a number of aspects. Names or aliases can be multiply used with varying structure, functionality and semantics.

Refinement abstraction is mainly about implementation and modularisation. It allows developers to selectively retain information about structures. Refinement abstraction is defined on the basis of the development cycle (refinement of implementations). It refines, summarises and views conceptualizations, hides or encapsulates details or manages collections of versions. Each refinement step transforms a schema to a schema of finer granularity. Refinement abstraction may be modelled by refinement theory and infomorphisms.

Encapsulation removes internal aspects and concentrates on interface components. Blackbox or graybox approaches hide all aspects of the objects under consideration. Partial visibility may be supported by modularisation concepts. Hiding supports differentiation of concepts into public, private (with the possibility to be visible to ‘friends’) and protected (with visibility to subconcepts). It is possible to define a number of visibility conceptualizations based in inflection. Inflection is used for the injection of combinable views into the given view, for tailoring, ordering and restructuring of views, and for enhancement of views by database functionality. Behavioral transparency is supported by the glassbox approach. Security views are based on hiding. Versioning allows developers to manage a number of concepts which can be considered to be versions of each other.

CROSS REFERENCE
I. DATABASE FUNDAMENTALS
   a. Entity-Relationship Model, Extended Entity-Relationship Model, Object Data Models,
      Object Role Modeling, Unified Modeling Language

REFERENCES