Collaboration

$3C = C^3 = \text{Communication} + \text{Coordination} + \text{Coordination}$

The Kiel Approach to Collaborative Systems

Collection of Recent Papers

Bernhard Thalheim

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

1 Table of contents

1. Development of Collaboration Frameworks for Distributed Web Information Systems
   published in [ST07]
   related papers: [Tha07a,Tha08,ST06,ST04a,ST05,FRT05]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008
2. Application development based on database components published in [Tha04]
   related papers: [ST06,BST06,Tha02b,Tha03]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008
3. SeSAM - support on profile, portfolio, and demand for (e-)parliamentarians
   published in [FKT04]
   related papers: [ST06,BST06,Tha02b,Tha04,Tha03]
   see also my website under interaction research
4. Towards Semantic Wikis: Modelling Intensions, Topics, and Origin in Content Management Systems
   published in [FT08] and [FT09] (final version)
   related papers: [JTK+08,FFRT06,FCF+]06,KBF+]05]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008
5. Facets of Media Types
   published in [ST08b]
   related papers: [Tha00,ST04b,MSTZ04,ST08a]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008
6. The Enhanced Entity-Relationship Model
   published in [Tha09]
   related papers: [Tha00,Tha07a,ST08c,ST08a,Tha07b,DMT07,MDT04]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008
7. Generalisation and specialisation
   published in [Tha09]
   related papers: [Tha00,BT07]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008
8. Abstraction
   published in [Tha09]
   related papers: [Tha00]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008
2 The Kiel Approach to Collaboration and Collaborative Systems

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.

2.1 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 modelling 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.

2.2 New Paradigms Raised by Collaborating Communities

Collaboration requires a change in computing paradigms beyond programming that can be based on Hoare rule semantics [ACKM04]. 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 [DT01] storyboarding language instead. It provides different conditions for steps such as accept-on conditions [Sri00] 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 modelled as collaborating components [Tha02a]. 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 [Loc03]. 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.

2.3 User Participation in Collaborating Communities

User participation in collaboration typically follows a general workflow that consists of a formation phase of groups or societies, in a working phase of collaborations and in a result of the collaboration. Any user in a collaboration can be considered at the micro, meso or macro levels. The micro level characterisation concentrates on the effects to a singleton user. The meso level is concerned with the groups and the impact the group has on its members or the member has on its groups. The Web forms regional, transregional or worldwide societies. These societies have an impact on the user development, on the collaboration and on the groups within these societies.

We sketch this differentiation in Figure 1.

Fig. 1. Micro, Meso and Macro Levels of Users Work
The separation of collaboration into these three main workflows also allows to specify the impact of collaboration. The impact to the singleton user is mainly self-presentation and affirmation in the first workflow, self-realisation in the collaboration workflow and well-being and satisfaction of information demand in the third phase. The impact is different for groups. It is either formation of intergroup relationships, playout of group stories and interrelations, or maintenance of cohesiveness at the third workflow. Similarly, we can distinguish the impact to societies: development of size or power for groups. It is either formation of intergroup relationships, playout of group stories and interrelations, or maintenance of collaboration workflow and well-being and satisfaction of information demand in the third phase. The impact is different for groups. It is either formation of intergroup relationships, playout of group stories and interrelations, or maintenance of cohesiveness at the third workflow. Similarly, we can distinguish the impact to societies: development of size or power for groups. It is either formation of intergroup relationships, playout of group stories and interrelations, or maintenance of well-organised societies based on policies such as XYZ-crazy, and social integration and participation afterwards.

References


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 be 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.

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.

![Figure 1: The collaboration triangle relating communication, coordination, and cooperation](image-url)
2.2 The Collaboration Style and Pattern

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 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, interceptor, extension interface), event processing (reactor, projector, asynchronous completion token, accept connector), synchronization (scoped locking, stratified 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 profiles.
- **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 presupposed for solution)
  - Instruments: (list of instruments for solution)
  - Collaboration: (collaboration style/pattern)
  - Auxiliary:
    - 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 □ (choice), || (parallel execution), ; (sequential ex-
execution), and \((.)^*\) (iteration). We may derive extended operations such as simple iteration \((.)^+\) and optional execution \(\Box \text{skip}\). 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:

\[
T; \ ((D \Box (C ; P)) \parallel (I ; U)) ; \ H ; (R ; H)^* ; \\
A; ((S \Box \text{skip}) ; E ; I ; N ; H ; (R ; H)^* ; A)^* ; S \Downarrow \\
\]

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 \(\nearrow\) and \(\searrow\) 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:** (obligations, permissions)

**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.,

\[
S = (M, \text{Man}, C, F).
\]

Media types offer their own functions including statistical packages, functions proposed for data warehouses, or data mining algorithms. The services manager $\text{Man}$ 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.

### 3.2 Exchange Frames

Exchange frames might be specified through the triple (Architecture, Collaboration Style, CollaborationPattern).

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 parties, their responsibilities, their rights and the protocols they may rely on.

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.

Beside the classical distributed system we support also other architecture such as database farms, incremental information system societies and cooperating information systems. Incremental information system societies are the basis for facility management systems. Simple incremental information systems are data warehouses and content management systems.

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

### 3.3 Collaboration Architectures

We observe that the three perspectives have a certain technical dependence. Collaboration must be based on communication. It follows rules of coordination. Finally, the top level of collaboration is the cooperation. With this layering we derive directly a technical structuring and layering of collaboration systems displayed in Figure 2.

---

**Table:**

<table>
<thead>
<tr>
<th>Party:</th>
<th>(names)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characteristics:</td>
<td>...</td>
</tr>
<tr>
<td>Profile:</td>
<td>...</td>
</tr>
<tr>
<td>Roles:</td>
<td>...</td>
</tr>
<tr>
<td>Rights:</td>
<td>...</td>
</tr>
<tr>
<td>Relations:</td>
<td>...</td>
</tr>
<tr>
<td>Part:</td>
<td>(general description)</td>
</tr>
<tr>
<td>Organization:</td>
<td>(general description)</td>
</tr>
<tr>
<td>Infrastructure:</td>
<td>(general description)</td>
</tr>
</tbody>
</table>

Parties are usually organized within organizations as groups:

<table>
<thead>
<tr>
<th>Organization:</th>
<th>(name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronization:</td>
<td>...</td>
</tr>
<tr>
<td>Stories:</td>
<td>...</td>
</tr>
<tr>
<td>Hierarchy:</td>
<td>...</td>
</tr>
<tr>
<td>Time slot:</td>
<td>...</td>
</tr>
<tr>
<td>Task distribution:</td>
<td>...</td>
</tr>
<tr>
<td>Coordination:</td>
<td>name</td>
</tr>
</tbody>
</table>

Infrastructures: (names)

Party profiles simply use the frame:

<table>
<thead>
<tr>
<th>Party profile:</th>
<th>(party profile name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information demand:</td>
<td>(general description)</td>
</tr>
<tr>
<td>Utilization pattern:</td>
<td>(general description)</td>
</tr>
<tr>
<td>Specific utilization:</td>
<td>(general description)</td>
</tr>
<tr>
<td>Party context:</td>
<td>(general description)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service:</th>
<th>(name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Based on:</td>
<td>general conditions</td>
</tr>
<tr>
<td>Media types:</td>
<td>(general description)</td>
</tr>
<tr>
<td>Raw media type:</td>
<td>...</td>
</tr>
<tr>
<td>Extensions:</td>
<td>...</td>
</tr>
<tr>
<td>Unit:</td>
<td>...</td>
</tr>
<tr>
<td>Order:</td>
<td>...</td>
</tr>
<tr>
<td>Co-/Adhesion:</td>
<td>...</td>
</tr>
<tr>
<td>Hierarchy:</td>
<td>...</td>
</tr>
<tr>
<td>Playout:</td>
<td>...</td>
</tr>
<tr>
<td>Services manager:</td>
<td>(general description)</td>
</tr>
<tr>
<td>Kind:</td>
<td>...</td>
</tr>
<tr>
<td>Communication:</td>
<td>...</td>
</tr>
<tr>
<td>Coordination:</td>
<td>...</td>
</tr>
<tr>
<td>Cooperation:</td>
<td>...</td>
</tr>
<tr>
<td>Competence:</td>
<td>(general description)</td>
</tr>
<tr>
<td>Task:</td>
<td>...</td>
</tr>
<tr>
<td>QoS:</td>
<td>...</td>
</tr>
</tbody>
</table>

The context of a service is characterized as follows:

<table>
<thead>
<tr>
<th>Context:</th>
<th>(name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media types:</td>
<td>...</td>
</tr>
<tr>
<td>Environment:</td>
<td>...</td>
</tr>
<tr>
<td>Range of variation:</td>
<td>...</td>
</tr>
</tbody>
</table>
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 parliamentarians 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
Application Development Based On Database Components

Bernhard Thalheim
Computer Science Institute, Christian-Albrechts University Kiel,
Olshausenstrasse 40, 24098 Kiel, Germany
Email thalheim@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 Introduction

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, difficult to maintain and to extend and 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.

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 modeling such as modular design by units[Tha00a] 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 modeling 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 [Tha00a] 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 modeling by specification of functionality and dynamic integrity constraints and interactivity modeling by assigning views to activities of actors in the corresponding dialogue steps. Co-design, thus, is based on the specification of 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.

An e-Government Application.

Modern parliamentary work has to face some problems due to the complexity of decision-
making processes. Citizens as well as politicians are often lost in the variety of topics that are discussed; they are lost in complex and dynamic political structures and hidden responsibilities. The SeSAM e-Democracy portal provides an easy-to-use collaboration and communication platform for politicians and interested citizens. Information is presented always up-to-date in all aspects with respect to the users specific needs and wishes while the system itself is easy to maintain and adaptable to local peculiarities. The information space of e-Government applications is rather complex. In Figure 1 we sketch this space.

Parliamentary systems provide information about persons, parties, parliamentary groups, and committees involved in the decision-making process, offer information about actual meetings, supports orchestration and syndication of complex documents (with older versions, annotations, protocols, or references to other related documents), and assists in collaboration, supports message exchange, suggestion making, sending criticism or opinions to responsible persons or groups. Such systems tend to be very large, become unmaintainable, are not extensible, and are finally not supporting but disturbing work.

We have analyzed systems available on the market and developed a database structure supporting the system. The outcome has been shocking. The initial schema used more than 800 relation types. Most of the relations are extensively interrelated to others. Extensibility, maintenance, development costs, integrity and semantics maintenance, change in technology, change of requirements becomes a nightmare for such systems. Experience gained in other projects have shown that the final system triples the complexity observed already for the in initial system.

For this reason, we decided to apply a new technique: component-based system development. Based on this technique, the SeSAM system system has been developed, is installed and used and can be step-by-step extended. The overall size of the final system is not smaller than the size of a monolithic system. But the system becomes maintainable, extensible, developable by smaller teams, and allows change of requirements and in technology.
Survey on the Paper.
We introduce the theoretical concepts of component-based co-design of database applications in the next section. Section 3 uses these constructs for development of methods for component integration that is non-invasive. We applied our approach to component construction in a number of projects, e.g., to an internet site that supports communication of parliamentarians in their daily work with documents. In the last section we demonstrate the power of our approach by discussing the document management in the SeSAM system.

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) [Tha00a] 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 $t = B | t \times ... t | \{t \} | [t] | l : t$,
- 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 $D = (S_1, ..., S_m, \Sigma_G)$ 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 $O^V_I$ and modification operations $O^V_M$. Based on this classification we derive an output view $O^V$ of $\mathcal{V}$ and an input view $I^V$ of $\mathcal{V}$.

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 [Tha00a].

**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 $S^K$ of the database with a set of corresponding input views $I^K$ and a set of corresponding output views $O^K$. 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 $\text{type} : C \rightarrow V$ for the view schemata $V$. 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 $\mathcal{K} = (S^K, I^K, O^K, S^K_C, \Delta^K)$ is specified by

(static) schema $S^K$ describing the database schema of $\mathcal{K}$,

syntactic interface providing names (structures, functions) with parameters and database structure for $S^K_C$ and $I^K, O^K$,

behavior relating the $I^K, O^K$ (view) channels

$$\Delta^K : (S^K_C \times (I^K \rightarrow M^*)) \rightarrow \mathcal{P}(S^K_C \times (O^K \rightarrow M^*))$$

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 $\mathcal{K}_1 = (S_1, I^K_1, O^K_1, S^K_C_1, \Delta_1)$ and $\mathcal{K}_2 = (S_2, I^K_2, O^K_2, S^K_C_2, \Delta_2)$ 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 $\mathcal{K}_1 = (S_1, I^K_1, O^K_1, S^K_C_1, \Delta_1)$ and $\mathcal{K}_2 = (S_2, I^K_2, O^K_2, S^K_C_2, \Delta_2)$ are called domain-compatible if $\text{dom}((\text{type}(C_1)) = \text{dom}((\text{type}(C_2)))$.

An output $O^K_1$ of the component $\mathcal{K}_1$ is domain-compatible with an input $I^K_2$ of the component $\mathcal{K}_2$ if $\text{dom}((\text{type}(O^K_1)) \subseteq \text{dom}((\text{type}(I^K_2)))$

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.

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 Document such as ApplicableFormat and LawsForSignature, and on subtypes which are used for additional properties such as Contract and Usage/Utilization in Figure 5. These additional properties are clustered according to their occurrence for the things under consideration. Furthermore, documents are classified by a set of
Figure 2: The Composition of Database Components

*meta-properties.* Finally, documents may have their life and usage cycle, e.g., *language versions.* Therefore, we observe that the schema is in our case a schema with 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., $\bigcup_{i=1}^{s} K_i = \{C_1, ..., C_k\}$ and $K_i \rightarrow C_0$, $C_0 \rightarrow K_i$ for $1 \leq i \leq s$ and $\text{card}(C_0, C_i) = (1, n)$ for $(1 \leq i \leq k)$.
- the extension types $C_{k+1}, ..., C_m$ satisfying the (general) cardinality constraint $\text{card}(C_0, C_j) = (0, 1)$ for $((k+1) \leq i \leq n)$.

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 schemata:

- Representation based on an entity type with attributes $C_1, ..., C_k$ and $C_{k+1}, ..., C_l$ and specializations 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 specializations 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 Raw Document - Operational Document - Blueprint - Submission - Approved Document - Archived Document cycle.

3 Non-Invasive Conceptual Construction Based on Components

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 modularization, parameterizability and conformance to standards.

We derive now a none-invasive construction approach which does not change components used for construction. Due to this restriction we gain a number of properties such as adaptativity, seamless gluing, extensibility, aspect separation, scalability, and metamodeling 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 \( \mathcal{K} = \{K_1, ..., K_m\} \) and labels \( \mathcal{L} = \{L_1, ..., L_n\} \) with \( n \geq m \). Given furthermore a total function \( \tau : \mathcal{L} \rightarrow \mathcal{K} \) used for assigning roles to components in harnesses. The triple \((\mathcal{K}, \mathcal{L}, \tau)\) 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.

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 \textit{harness} consists of the harness skeleton \(\mathfrak{H} = (\mathfrak{K}, \mathcal{L}, \tau)\) and the harness filter \(\mathfrak{F} = \{(L_i, \mathcal{V}^L_i) \mid 1 \leq i \leq n, L_i \in \mathcal{L}, \mathcal{V}^L_i \subseteq \mathcal{V}_\tau(L_i)\}\) for a set of wrapped components \((\mathcal{K}_i, \mathcal{V}_i)\).
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 $H_1 = (K_1, L_1, \tau_1)$ and $H_2 = (K_2, L_2, \tau_2)$ 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 $\{1, ..., n\}$ such that $K_{\tau_1(i)} = K_{\tau_2(\rho(i))}$. The bulk harness of unifiable harnesses $H_1, ..., H_p$ is constructed by renaming the labels $L_j$ of each harness $H_i$ to $L_{i,j}$ and combining the label functions $\tau_i$.

**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*.

![Figure 4: The Waffle Architecture Composition](image-url)
4 Using the Power of Component Construction for Management of Administrative Documents

The SeSAM System.
The SeSAM system (system for simple storage, retrieval and modification of internet content) has been developed as an extension of the internet platform www.cottbus.de. It supports the work of parliamentarians, their internal and external communication and their work with documents. Information is displayed to visitors of the website depending on their role, the content available, the task currently under consideration, the associations within the content set, and the technical environment. Content of the website is determined by the schedule of parliamentary sessions, on groups in the parliamentary, on the state of the document processing, and discussion forums currently open.

SeSAM is a content management system. It supports the export and the import of documents, news, and contributions. Data are sorted into folders and projects. Documents have a number of versions. SeSAM has sub-systems for accounting, configuration, archiving, and oral communication. Within this paper we concentrate on the document management component of the system.

SeSAM components have been also integrated into the inhabitant information service of www.cottbus.de. Inhabitants are supported by download facilities for documents, by intelligent search facilities for document retrieval depending on their daily needs, by communication facilities for direct communication with the city administration and by adaptation of the presentation to their current technical environment such as channel, display equipment and personal profile.

Document Characterization.
Documents in administrations are very complex. They evolve over lifetime. We can distinguish different phases:

**Raw document:** Raw documents or frames or prototypes are document forms with mostly empty data, with emphasis and default values. During call of a document by a department, the document is instantiated by the default values associated with the department or unit calling the document. Instantiation can be also performed by computing queries on the document base, especially on archived documents. Components or attributes to be instantiated during the call of the raw document are called **configuration parameters**.

**Operational document:** The entry of data into the document may be a rather complex workflow with several stages, different associated actors, various responsibilities and different stages of preparations.

**Blueprint:** The document is almost ready. The content which is required for a filled document exists. The documents needs a number of approvals in order to be submitted.

**Submission:** The document to be submitted has the complete content to be required by the business processes and rules. It is used in the working process. The document may be changed. In this case, the document is pushed to the state blueprint.
Archived document: The document or parts of the documents or views on the document are stored in the archive. Archived documents have a summary or docket [SS99]. They cannot be changed. They can be retrieved.

These lifespan phases are supported by the lifespan composition of document star schemes and use incremental composition.

Documents may have a complex structuring which is partially displayed in Figure 5. We, however, assume:

**Inductive construction:** Each document can be constructed on the basis of simpler documents and base documents by application of constructors.

This assumption restricts the application areas. However, based on this assumption we are able to compose the document step by step. In this case, structural recursion can be applied to construction of necessary functionality.

The semantics changes dynamically. Thus, we are not able to specify the entire document by means of static integrity constraints. In order to model the change of semantics we introduce the concept of states. We associate documents with their state thus providing a version control. The actual document is generated by unfolding the versions.

Version control may be rather complex. We can use the concepts of the life space of a document which is similar to the concept of the story space:

**Life flow of operational documents** is defined by a rooted graph \( G = (N, E) \) which nodes reflect the variety of states a document may have and which edges represent the actions causing a change of the document.

**Nodes** have a unique identity and represent the delta of the change to the document imposed by the action which is applied to the predecessor.
Directed edges are associating two nodes. They have a label that is used for representation of action kind, action time, actor, role, rights etc.

Construction of documents is based on the concept of snapshots and on cuts or stratums in the life flow graph: At a given moment of time \( t \) and for a given document the graph is captured by unwinding the document graph:

- Those nodes \( N_t \) are captured which time stamp is smaller than the given point of time.
- The subgraph \( G_t = (N_t, E \cap (N_t \times N_t)) \) is used by iterative construction of the document beginning from the root of the graph.

Construction may lead to documents which have a number of optional components. The graph \( G_t \) is thus used to generate a snapshot \( D_t \) of the document \( D \).

Further, given a group of users \( U \). A view of the document is based on the subgraph \( G_{t,U} \) which uses only those edges which visibility rights belong to the group \( U \) or are assigned to ‘public’. Only those nodes are used for the delta capturing which are reachable from the root.

The construction mechanism may also be used for keeping the current version of a document. The current version describes the latest state.

Supporting Functionality for Documents.
The construction is performed by the

ICE: Information and Content Extractor

which has been developed for runtime and context-sensitive adaptation of the database design systems RADD and ID\(^2\) to the designers [Tha00a]. A similar application is the adaptation of web search tools. The ICE is based on the following concepts:

Built-in abstraction for data for coarsing/zooming, indexing, viewing, and aggregation of documents enable in surveying and landscaping documents.

Farms of views can be inductively and incrementally generated with or without materialization and with or without recharging after modifications.

Just-in-time views on the basis of materialized or prefetched views support quick response by right document, at the right place, at the right time, in the right condition, and at the right cost.

Parametric views have parameters. A special group of parameters are the context parameters which enable in adaptation to the technical context (infrastructure, channel), to the task-ruled context (workflow, history), and to the actor context (profile, role).

Runtime versioning supports the management of views currently in use. These views disappear (‘fade’) whenever they are not needed. The handling is based on garbage collection procedures.

We notice that ICE has been also used in one of our industrial project for the generation of bulk workflows. This concept is very similar to the discussed above document concept. It is a useful concept for e-Learning sites. Bulk workflows support the typical parallel workflow in offices:
• Work is carried out in a high parallel form.
• The flow of tasks requires feedback and support of waiting queues.
• Work is often controlled from outside.

Bulk workflows are based on

**unfolding of general workflows** with a generic runtime workflow,

**potentially high parallelism** of workflows with intermediate states, to-do lists and resource sharing,

**multiple-choice workflows** for roles, free choice and bundling of tasks, and

**transactions-based meta-workflows** with management of resources and roles.

The operations we use for the change in documents are based on **paragraph anchors**. Paragraph anchors are automatically generated together with the operations applied to the document. We use the following elementary operations:

**Insertion** of new components $c$ into the document at a point $p$ assigned. Given a document identifier $d$, a node $n$ in the life flow, and a moment of time. Then, an actor $a$ which has the right $r^*$ to change the document in a role $r$ causes a complex insert at a the point $p$:

$$
\text{insert}(d,n,c,p,t) = \text{insert}(G, \{n'(new, c), (n, l(add, t, a, r, r^*, ..., n'))\});
\text{update}(G, n, add(p)) .
$$

**Deletion** of anchored paragraphs $p$ from the document identified by $d$ in a node $n$ in the life flow at a moment of time $t$ by an actor which has the right $r^*$ to change the document in a role $r$ is defined by:

$$
\text{delete}(d,n,p,t) = \text{insert}(G, \{n'(new, \lambda), (n, l(erase, t, a, r, r^*, ..., n'))\});
\text{update}(G, n, add(p)) .
$$

**Updating** of anchored paragraphs $p$ by a component $c$ in the document identified by $d$ in a node $n$ in the life flow at a moment of time $t$ by an actor which has the right $r^*$ to change the document in a role $r$ is defined by:

$$
\text{update}(d,n,c,p,t) = \text{insert}(G, \{n'(new, c), (n, l(replace, t, a, r, r^*, ..., n'))\});
\text{update}(G, n, add(p)) .
$$

A modification of the document is a sequence of elementary operations. The modification forms a sub-graph. If the modification is a sequence we can condense the path to one edge between the end node and the starting node.

**Strongly sequenced documents** allow a simpler handling. If only one actor is acting at a certain point of time and all actions are only applicable to the most recent state then the life flow is a queue of actions applied. In this case we may store the most recent version of the document and invert the life flow to a history flow. We find, however, that due to a variety of rights and roles online documents cannot be modeled by strongly sequenced documents.

The **life flow of operational documents** corresponds to **construction of documents**. There is a large number of document management system [KMF98] such as EDMSuite (IBM),
Domino.Doc (Lotos), ImagePlus. Different standardization committees such as the ODMA (open document management API) and DMA (document management alliance) have been working on a standard. The problem of all these proposals is, however, that none of them was able to capture the complex structuring of a document's life flow. Our approach allows the integration of life flow, content of documents and workflow.

Varying Documents for Play-Out.
Documents have a large variety of representation views:

**Actor view:** An actor or a group of actors have their own views based on the access rights that are valid for the actor.

**Role view:** A document can be used in a number of roles. These roles are supported by a variety of partial views on the document.

**Function view:** Documents are often instruments for complex business processes. The specific content necessary for the document in the business process is generated as a function view.

**Task view:** Documents are used to support tasks. These tasks require the insertion of certain data into the document. The task view consists thus of a representation similar to a printed form.

**Node view:** The responsibility of the content for a document is often distributed among a set of users. A document may have parts for which a user plays the role master. The copies on other nodes have the role slave. Slaves do not have the right for updates or other modifications.

Documents can be represented by snowflakes. They have a variety of representations.

**Configuration of a document:** As already described the construction of documents can be based on the life flow. The snapshot at a certain moment of time displays the state of the document.

**Run-time adaptation depending on the state:** Concepts used for the representation of documents are adhesive and cohesive to each other in a different manner. If a component currently under consideration is less cohesive to some of the components than these components may be cut out during consideration of the component currently under consideration.

This representation structuring of documents leads to the conception of recursive folders of documents. If the structure is defined on the basis of a star sub-schema and the life flow is representable by a snowflake then the variety of views is representable by a multi-list. The different branches of the multi-list are opened at different moments of time.

Summarizing we can model the document by a multi-dimensional structuring:

**Intext:** The intext of a document follows the internal logical structure of the document itself, is used for the main characterization of the document, and carries the specialization hierarchies.
Context-dependent representation: The context of the document is a manifold of associations to other documents, the temporal and spatial relations, the application context, the interface context, and the meta-characterizations of the document.

Version under consideration: The document version, the archive version, the log version and the view versions of the document are combined in the version space of the document.

Life flow of the document: The evolution of the document and the changes imposed by a variety of actors are represented by the life flow of the document.

The four dimensions are used for run-time configuration of documents. Documents are classified by meta-parameters. These meta-parameters are dimensions. One of these meta-parameters is the applicable overlay structure:

- The presentation forms are represented by parameters enabling in adaptation of presentations. Web applications use the full-screen modus of representation although most of users try to use web browsers in a half-screen modus. We may have also small information containers [FST98]. Therefore, the representation of documents is supported by functions for generating display options, for decomposing documents and thus generating junks of the same document, for adaptation to the size of the window, for enumeration, for cutting pages of the document, for creating page turners and passages.

- User-defined overlay structures use anchors provided by the document structure. The anchoring is usually hidden from the user. The anchors may be used to create user-specific bookmarking. Bookmarks may be of arbitrary graph structure. A variety of ordering and presentation functions can be applied. Anchors have their own name space which consists of the identification of the document and the internal anchor name. These anchors can be combined with the index or abstraction facilities.

Specific overlay information is made in form of annotations. Annotations can be related to the content or to the functionality.

*Thumb nails* are specific bookmarks. They support scanning and zapping through documents.

- Documents are often formulars. Therefore, a user-defined version is instantiating some sections, paragraphs or forms provided by the document. Since these instantiations relate the document also to other data sources, an update functions is included into the form insert procedures. Forms may be also added to the document as well as specific appendices. These functions must be reversible. Forms may be included into the representation or may not.

- Documents may be combined or associated by the user. The bundling mechanism enables in combining sets of documents by the user. This mechanism generalizes the folder concept. We do not store, however, the same document in different folders but mirror the documents in a master-slave mode. Whenever, we need the full document the master document is retrieved. Whenever a change is made then the master document is used. Whenever we need a user-defined document then the master document is used together with the slave that is bound to the user or to the workflow.
• Documents are characterized by keywords, topics or ontology items. The characterization may be applicable to entire documents or to anchors in documents. The ontology characterization of documents supports intelligent search especially associative search.

• Documents often use specific menu functions. These functions are added to the header of the document. They enable in variable display, in applying the provided functionality and in export of the content.

• Finally, dockets\textsuperscript{99} of documents are specific overlay structures. Dockets provide information
  - on the content (abstracts or summaries),
  - on the delivery instruction,
  - on the parameters of functions for opening the document (opening with(out) zooming, breath, size, activation modus for multimedia components etc.)
  - on the tight association to other documents (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 control policies,
  - on applicable workflows and the current status of completion and
  - on the receipt of the document which enable in tracing the document life cycle.

The overlay structuring of documents is supported by the onion approach\cite{TD01}. Onion generation is based on layering that used for generation of website functionality and content. The onion approach nicely fits with the translational approach. It is our aim to generate consistent sets of associated XML documents. Let $\mathcal{X}$ be the set of all XML documents under consideration. Further given a generation algorithm $G$ applicable to XML documents that allows to generate new XML documents on the basis of the given ones. Let us denote by $[\mathcal{X}]_G$ the transitive closure of the generation algorithm applied to $\mathcal{X}$. A set $\mathcal{X}$ of XML documents is called consistent according to $G$ if all inner references in $\mathcal{X}$ belong to $[\mathcal{X}]_G$, i.e., no dangling inner references are in the set $\mathcal{X}$. In this case $\mathcal{X}$ is called XML suite.

Document systems should be supported by a specific data warehouse architecture:

**Play-out servers** present, store and protect released content. The play-out of documents depends on their usage. Typical widely used documents are documents used in logistics:

• Bills have their own numbering and their own format. They serve also as an contract of carriage between shipper and carrier.

• Certificate on the content and the origin of the contents are used for statistical research, and for accessing duties, particularly under trade agreements.

• Invoices declare against which payment is made. They are used for clearing documents.
Dock receipts are issued by the forwarder on exporter’s behalf. They include shipment description, physical details, and shipping information.

Bills of lading are used as contracts between carrier and shipper, spell out legal responsibilities and liability limits for all parties to the shipment.

Packing lists provide details on the packing procedure of the container.

Sight, time drafts instruct the buyer’s bank to collect payment.

Production servers have controlled access to documents and host dockets.

Specific docket servers manage trusted content exchange between the servers.

Generic docket servers communicate and encapsulate value-adding services.

5 Conclusion and Outlook

Component construction has been widely used in other engineering areas. Database modeling is still based on handicraft approaches, i.e., each new application is developed from scratch or uses solutions which are again based on handicraft. Database systems become now part of middleware solutions. Thus, a plug-in approach must be developed that allows a stepwise integration into existing infrastructure. Component construction supports a plug-in approach.

This paper has introduced a general approach to component construction. We observe a number of advantages of component construction such as: simpler sub-schemata, simpler combination, simpler integrity maintenance, modularity and extensibility. Component construction does not lead, in general, to optimal schemata defined on the number of types. Handicraft approaches may lead to such schemata. However, the advantages will play an important role whenever the application is complex and the database schema becomes large.

We defined harnesses and harness construction. This mechanism allows to construct information systems based on components which are associated to each other by hyper-edge wires of harnesses. We have applied this approach in the SeSAM project. The application led to simpler schema construction and led to a schema that is easy to extend, to query and to maintain.

Component construction may be extended by construction with component functionality. The development of a theory of component functionality is our main research direction in future.

References


Remark: It is not my aim to discuss related database and internet technology. Our group uses the website modeling language SiteLang [TD01], the power of advanced object-oriented approaches [ST98], the theory of interaction [Tha00b], and the co-design approach [Tha00a, Tha03a].

I thank the EJC’2004 reviewers of the paper for their substantial suggestions for improvement of this paper.
ABSTRACT

Modern parliamentary work has to face some problems due to the complexity of decision-making processes. Citizens as well as politicians are often lost in the variety of topics that are discussed; they are lost in complex and dynamic political structures and hidden responsibilities.

In this paper we present the SeSAM e-Democracy portal which provides an easy-to-use collaboration and communication platform for politicians and interested citizens. Information is presented always up-to-date in all aspects with respect to the user’s specific needs and wishes while the system itself is easy to maintain and adaptable to local peculiarities.

After analyzing the needs and requirements of an e-Democracy portal to be widely accepted by politicians, citizens and members of municipal administrations we will discuss the functionality of SeSAM as well as the methodological background of user-adaptive information systems, like database schema pattern for documents and collaboration, profiling, system adaptation and flexible information play-out.

KEYWORDS

e-Government, e-Democracy, collaboration, profile, adaptation, pattern

1. INTRODUCTION

To speak about e-Government means to speak about two different types of private-public interaction: e-Administration and e-Democracy. E-Administration covers all tasks providing access to public administrative services, e.g. applying for an identity card or filing a tax return. On the other side e-Democracy describes the electronic participation of citizens in political decisions as well as the interaction between politicians and inhabitants.

The comparison between the Internet presentations of 82 cities in Germany with more than 100,000 inhabitants ([I21]) shows that the central point of action in e-governmental related issues is located in the area of e-Administration because of economic reasons. Elements for public participation are present but still widely extendable.

The “Balanced E-Government” study published by the Bertelsmann foundation ([BF02]) proposes an integration of administrative and participative elements in municipal e-Government portals. Following this approach we want to present the SeSAM e-Democracy portal providing information about parliamentary activities as well as interaction facilities for an easy-to-use collaboration between politicians and citizens.
This application uses the approach of providing information depending on profile, portfolio, and demand of the user. We discuss the organizational and methodological background needed for such an application to be widely accepted by citizens as well as politicians.

Currently, SeSAM works as the e-Democracy part in the Internet presentation for a German city with approximately 100,000 inhabitants.

2. PROBLEMS OF MODERN DEMOCRACY

People involved in modern parliamentary work are faced with a couple of problems which can be described by the keywords “awareness”, “informedness”, “last-minute collaboration”, and “complexity”.

For taking part in political decision-making processes the interested citizen has to be informed about the structure of the parliament and its institutions. He wants to know which people are member of parliament or a certain committee, which parties or parliamentary groups are present, who is the head of a committee or who is responsible for a certain task. Finally, for a successful collaboration the citizen needs additional contact information.

There is no question that all of this information can be put on a simple web page. Looking at web sites providing structural information about political institutions shows that this is still a common approach. But most of this information is quite dynamic, e.g. some members leave the parliament and new ones move up, parties change their collaboration in parliamentary groups or new committees are set up. Keeping track of these changes and providing them to the citizen by a static web site is a challenging task because changes may affect other dependent information as well.

Beside structural information about the political institutions citizens as well as politicians need to be informed about the tasks the parliament or committee is talking about. Regularly, there exist dozens of bills and motions in parallel that have to be discussed. Additionally, these documents exist in different versions from different stages of development. It is hard for a politician to keep up-to-date in every single discussion on every single document. During their sittings parliamentarians spend a lot of time by searching the appropriate document in two or three thick folders containing all available records. For citizens and visitors that are not directly involved in the decision-making process it is even harder to follow the discussions.

Another problem is the complexity of documents due to the complexity of workflows within the democratic processes. Bills and motions have to go through a complex network of institutions and decision-makers to become a regulation. These workflows are grown through the years and highly specific to the content of the related document. Most of these workflow steps produce new document versions, annotations, protocols of discussions, votes or press releases. Politicians and citizens want to become an overview over the state of a certain document. They are interested in involvements and responsibilities – information that is hard to receive for non-experts in municipal affairs.

All of the problems described above make it hard for politicians to interact with citizens or other interested persons. People expect that every question is answered in an appropriate amount of time – not more than a few hours or a few days. Every time – and sometimes everywhere – when a citizen is calling for a certain topic a politician needs up-to-date information about this topic in an overall way without loosing time by searching all of his folders. Results of these unofficial discussions should have an influence to the whole decision-making process like any other discussions do.

In the following sections we want to present how these problems can be faced by an Internet-based collaboration platform.

3. FUNCTIONAL ASPECTS OF SESAM

3.1. Requirements

Based on the analysis of the problems in modern democracy described in section 2 we formulated the following requirements for the SeSAM collaboration platform.
The first piece of functionality that the application has to provide is information about persons, parties, parliamentary groups, and committees involved in the decision-making process. This information includes relationships between these entities, e.g. which persons belong to which parties, as well as contact and background information.

The second part of the application has to offer information about actual meetings. Politicians (as well as citizens if these meetings are public) can access the associated agendas. Additionally, background information like references to related documents is provided. So the user has always access to all relevant documents for the meeting he wants to prepare for or he is sitting in.

By browsing the structure of a complex document the user can access older versions, annotations, protocols, or references to other related documents. For example, he can see which committees already discussed over this topic and which results were obtained.

As the fourth part collaboration facilities come into play. To simplify the interaction between politicians and between politicians and citizens the application has to provide some kind of communication management. The first form of supported communication is the possibility to publish messages to certain types of users. For example, a user can write messages for the party or the committee he is member of. Another example is the publication of press releases to inform about the work of the parliament.

The second type of communication serves for the reverse case. Citizens can write suggestions, criticism or opinions to responsible persons or groups. The information “Who is responsible for this task” can be directly taken from the structural data provided the application.

3.2. SeSAM Overview

As the overall principle the information system is designed for providing information by profile, portfolio, and demand of the user. The profile of the user includes his usage preferences as well as specific access permissions. The portfolio of a user describes the tasks this person has to fulfill by using the application. Providing information on demand assists the user to get the needed information just in time.

To achieve the suggested approach the management of structural information of political institutions plays a central role within SeSAM. For example, access checking and information play-out set up on the role of a user that he plays in the real political life.

Beside the role management SeSAM includes modules for managing meetings and appointments, documents like bills and motions, as well as messages for the communication between politicians and citizens. In the following sections we take a closer look at these modules.

3.3. Users, Roles and Groups

As mentioned before, the management of users according to their roles in the political life is the main source of information for all functions in SeSAM. Firstly, we distinguish three different types of users:

- **The anonymous user** represents all system users which have not provided any authentication information. Typically, this group of users contains citizens interested in local politics. That’s why the portfolio for this type of user contains mainly the play-out of public information or public documents. Anonymous users have no individual profile, only a generic one.

- **The registered user** was successfully authenticated by the system. This type of user represents politicians or members of the municipal administration with an existing profile and an associated portfolio of individual tasks.

- **The administrative users** are registered users with special rights, e.g. adding new users, creating new parties or committees, or assembling new agendas. Although there exists a system administrator, every group can have its own local administrators for certain local tasks. There is no need to grant overall access privileges to a single person. Administrative permissions can be mapped to user roles corresponding to the responsibilities and tasks in real life.

The role of a registered user within the political institutions is directly taken in consideration. For example, if a user is elected to a certain committee, he has automatically access to all documents and messages for this group. In the reverse way the system presents the user only these documents and messages which are relevant for his memberships. Changing a membership of a person directly causes a change of the access rights of this person. Because the role of the user is the only way to define access permissions, the
System user administration can be restricted to tasks like creating a new user or removing him. The user's profile and portfolio is always kept up-to-date by a single change of the structural information – which has to be updated in that case for play-out in any way. Figure 1 shows a screenshot of the SeSAM role management.

![SeSAM Role Management](image1.png)

Figure 1: SeSAM Role Management

In the reverse case, the play-out of structural information is also role-based. The user can query for information on a specific role, e.g. “What’s the telephone number of the head of parliament?” The role “Head of Parliament” is associated with a certain person, so the contact information can easily be obtained. When the head of the parliament is reelected the user gets automatically the new telephone number.

To become familiar with current available groups and memberships the user can browse through the database, figure 2 shows an example.

![Browsing the Structure](image2.png)

Figure 2: Browsing the Structure

### 3.4. Meetings and Appointments

Official meetings in municipal parliaments are either public or not. Public meetings can be visited by interested citizens, so the agenda of such a meeting has to be announced. Commonly, this is done by an official message in the local press or the citizen can ask the city administration for a certain agenda. A typical agenda consists of several items while most of them are associated with a specific document like a bill, a
motion, or a question. While reading this agenda the typical user wants to become background information about the topics, so SeSAM supports the user with links to the documents an agenda item is relying on.

When the meeting is over, SeSAM keeps track of the discussion and provides a protocol with the results of the votes.

Because parliamentary meetings may have a public and a non-public part the agenda is played-out with respect to the user’s rights. Figure 3 show as an example the agenda of a meeting: on the left side from the citizen’s point of view, on the right side from the politician’s point of view.

![Figure 3: Information Play-Out Depending on the User’s Profile and Portfolio](image)

### 3.5. Bills, Motions and Workflows

Documents in public decision-making processes are processed by a complex network of responsibilities. For example, they are handled by different committees in parallel, can be rejected, or discussed for several times. The knowledge, which committee or institution has to handle which document at a certain point of time is determined by law. Unfortunately, these regulations are not well understandable, changing over time and are highly specific.

To enhance acceptance during the members of the city administration we decided to leave the knowledge about workflows outside the system in the responsibility of an experienced user. Normally, there exists such an institution in the city administration because these workflows have to be prepared and controlled in real life, too. This approach has two advantages: the members of the city administration do not have the impression of being replaced by software which results in a better acceptance. On the other hand the system is better adaptable to changing political environments.

The workflow in SeSAM is simplified to a distribution scenario. When a bill or another document is brought in, the distributor uses his personal knowledge to put the document on the agendas for the meetings of the responsible committees. After the meetings the distributor collects the protocols and annotations and creates new document versions. Knowing the next steps in the decision-making process he can now put the document on the next agenda.

Besides supporting the work of the distributor SeSAM allows any user depending on his rights to browse through the actual and archived documents.
3.6. Messages

In the first case, SeSAM messages are intended to work as announcements from politicians for politicians or citizens. Every registered user can write messages with different levels of visibility: public, internal or for a specific receiver. Public messages, e.g. press releases, can be read by any anonymous or registered user. Internal messages are visible to registered users only. Messages that were sent to a specific receiver group are only visible to the members of that group.

A user can write messages in the name of all groups he is member of. To ensure quality, every group can locally define a message administrator. This person has to review all messages coming from this group before they were published. Additionally, the message administrator decides which messages sent directly to his group are visible to the members or not.

The second way of communication support in SeSAM is sending messages from citizens to certain responsible institutions. For this reason, telephone numbers, email-addresses, etc. are provided as mentioned in section 3.3.

4. THE METHODOLOGICAL BACKGROUND OF SESAM

Building an application like SeSAM needs a well-suited systematical and theoretical background to keep the system flexible, extendable and adaptable. We want to discuss some aspects of information system design used to develop our collaboration platform: database schema pattern, profiling and adaptation support, and flexible information play-out.

4.1 Documents and Collaboration Pattern

Documents are complex data structures growing during their life time ([Tha01]). Looking at their political life cycle we can identify some common stages:

- **Documents in preparation**: Members of city administration as well as members of certain parliamentary groups prepare an official document as a request for discussion. This can be a very complex process but usually it is done without support by a collaboration platform like SeSAM. So we can assume that document creation is an atomic operation.

- **Documents waiting for discussion**: When a document is brought on its way the distributor mentioned in section 3.5 has to put the document on the agendas of the appropriate committees.

- **Documents in discussion**: Documents being on an agenda are processed by the related committee. The document can be accepted, rejected with remarks or completely rejected. After this discussion the distributor collects the protocols and annotations and creates new document versions. Normally, a document runs several times through the stages “waiting for discussion” and “in discussion”. Every time, different parts of the document are collected.

- **Archived documents**: When a document is finally accepted or rejected it will be kept in the database for historical reasons. Archived documents cannot be updated.

The semantic of a document may change during its lifetime. That’s why we decided to introduce the concept of states. Every state is associated with a document version. Every document version can have several preceding versions as well as several succeeding ones. On this point of view a document forms a rooted graph which nodes represent the versions of the document and which edges are associated with the action that caused the document’s change as well as permission specifications. Every version is associated with its specific elements like protocols or annotations. Retrieving a certain version of the document means to traverse the document’s graph and unfold all preceding versions.

Modeling such complex structures in a handicrafted way may lead to inflexible database schemas that are hard to maintain. For that reason the SeSAM database schema was designed following a pattern-based approach. Concepts are modeled by star and snowflake sub-schemata and plugged together on certain points of contact.

In SeSAM we use a simplified version of the generic document pattern that was proposed in [Tha01]. The document’s kernel type includes attributes of common interest, e.g. the title of the document, its creation date.
or an abstract. The kernel type is extended by a variety of subtypes describing certain additional aspects of
the document, like the author of the document or associated annotations. Some of these subtypes are bridge
types; they connect two different concepts with a special meaning. For example, the author plugs in the
document type as well as in the institution type.

For modeling persons and organizations we use the pattern described in [Tha00]. The concepts of persons
and organizations are connected by the membership relation. The kernel types contain attributes that are
specific to the associated concept while subtypes like the address or usage roles are common to both ones.
Because persons as well as organizations function as institutions, both concepts can be generalized.

The other data structures in SeSAM, e.g. agendas or messages, can be handled in a similar way by the
document pattern.

4.3 Profiling and System Adaptation

As mentioned above there exist a need to adapt the functionality of the application. The first kind of system
adaptation is the adjustment to local requirements and peculiarities. For example, there exist special
restrictions on data protection. Some administrations may want to hide internal telephone numbers.
Additionally, there exist different levels of visibility depending on the user’s role. This kind of filtering can
easily be solved by using views instead of operating directly on the tables of the database.

A second kind of adaptation is the extension of the schema. For example, some administrations want to
include the speaker’s notes in the discussion to the related document. Most of these extensions have only
local affects and are used within simple play-in/play-out scenarios. Together with flexible play-out facilities
the pattern-based design approach enables the application to simply plug dependent concepts into the core
concepts of SeSAM.

Additionally to system adaptation the application has to take care of user specific profiling issues. Every
user is associated with an individual profile and task portfolio. They are determined by the user’s
memberships in the different groups. Access rights are modeled by using the concept of user roles.
Permissions are associated with a user role, not with a specific person or group. As an additional layer of
indirection they separate structural reorganization from system administration. Depending on the user’s roles
information is played out or hidden, e.g. only members of a certain group can read the messages that were
sent to this group. In the same way, functionality is offered depending on the user’s role.

4.4 Play-out Facilities

The raw information from the database must be presented in a user specific way. As mentioned in section 4.3
the data has to be adapted to the general system requirements and to the profile and portfolio of the current
user. In a second dimension the user may have certain preferences, e.g. different media types for output.
Additionally, the information system may be embedded in a container application that provides a
“framework” for presentation.

To ensure a flexible step-wise information play-out, SeSAM uses the XML-based onion approach
introduced in [ThD01]. Figure 4 depicts the different layers of generation.

The data from the tables in the database is accessed by views to provide filtering and is formatted as a
XML document. In the next step this XML document is enhanced by navigational elements from the actual
usage context. This packet of information is integrated in the application context by adding the representation
of additional service functionality. In a last step the data is transformed due to user’s personal preferences or
style guides to produce the representation for the desired output media. These step-wise transformations are
handled by simple XSL transformations.

5. CONCLUSION

We presented the running e-Democracy portal SeSAM which enables politicians and citizens to use the
Internet for collaboration, communication and information. SeSAM meets the requirements directly
formulated by its users – politicians as well as citizens.
Because of its pattern-based database schema design, its role based profiling model and its XML-based play-out-facilities SeSAM is easy to use, easy to maintain, flexible, adaptable, and extendable.

ACKNOWLEDGEMENT

We like to thank Cornell Binder, Stefan Dieringer, Henry Fröschke, Thomas Kobienia, Sven Schoradt and Pierre Smits as well as all other members of the Info Services Team at the Brandenburg University of Technology at Cottbus not mentioned here by name. Special thanks go to Gabriele Bogacz from the city administration of Cottbus.

REFERENCES


Towards Semantic Wikis: Modelling Intensions, Topics, and Origin in Content Management Systems

Gunar Fiedler, Bernhard Thalheim
Department of Computer Science
Kiel University
Olshausenstr. 40, 24098 Kiel, Germany
{fiedler, thalheim}@is.informatik.uni-kiel.de

Abstract

Content management is the process of handling information within an organization or community. Therefore, content management systems have to provide generic functionality for generation, extraction, storage, and exchange of digital assets. Because of the heterogeneity and complexity of content, a sufficient semantical and user-oriented annotation of content is crucial. Although semantical annotation by metadata and ontologies together with reasoning support has been extensively studied for a long time, commercially available content management systems provide only basic support for semantic modelling. Conceptual aspects of content users and support of user specific intensions are neglected. In this paper we will analyze the mismatch between the requirements of content management and semantical description and propose a data model for content which treats semantic information not only as describing metadata but incorporates the data itself, the intension behind the data, the usage of data and the origin of data on the same level.

1. Introduction

Content in its actual definition is any kind of information that is shared within a community or organization. In difference to data in classical database systems content usually refers to aggregated macro data which is complex structured. Structuring of content can be distinguished:

- The structure of the aggregated micro data is preserved but micro data was combined to build larger chunks of information. Examples are scientific data sets such as time series of certain measurements. There is a common (or even individual) structuring and meaning for each sampling vector but the compound of all sampling vectors adds additional semantics.
- The structure of content is only partially known. A typical example is the content of Web pages: structuring is known up to a certain level of detail which may also be varying within one instance.
Content may be subsymbolic, such as pictures, videos, music or other multimedia content.

Aggregation of content usually takes place by combining reusable fragments provided by different sources in different formats such as texts, pictures, video streams or structured data from databases. Content is subject to a content life cycle which implies a persistent change process to the content available in a content management system (CMS).

Currently, many systems claim to be content management systems. A recent overview of the German market (www.contentmanager.de, viewed June 12th, 2007) reveals hundreds of products related to tasks of content management. Most products are related to Web content management. These products organize content for Web pages with a strong orientation on editorial components such as texts and pictures.

The more generic ones agree in a major paradigm: the separation of data management and presentation management. Data management reflects the process of supporting content creation, content structuring, content versioning, and content distribution while presentation management grabs the data for delivering it to the user in various ways. Only content which is generated following this separation can be easily shared, distributed, and reused.

Following new trends and developments in Web technologies, e.g., in the context of Web 2.0 or the Semantic Web the automated processing of content becomes more and more important. Because content represents valuable assets it may be reused in different contexts (content syndication) or has to remain accessible for a long time.

The semistructured or even unstructured nature of content requires annotations to enable search facilities for content. Expressing semantics in a machine interpretable way has been under investigation since the early days of artificial intelligence, see e.g., [12] for a survey of knowledge representation techniques such as logical theories, rule-based systems, frames or semantic nets. Today systems handle semantical descriptions as metadata describing certain content instances. There are different ways for associating data and metadata:

- A conceptual, logical, or physical schema is defined and instances are created according to this schema. This is the usual way for classical databases. The modelling language strongly restricts the capabilities of this description facility. Common languages such as Entity-Relationship Modelling or UML focus on structural properties with support of selected integrity constraints.

- Defining a schema is not applicable (or only in a restricted way) to semistructured or unstructured content. For that reason content instances are annotated. An annotation is a triple \((S, P, O)\) where \(S\) denotes the subject to be annotated, \(P\) a predicate denoting the role or purpose of this annotation, and \(O\) the object (or resource) which is associated with \(S\). The vocabulary for annotations is organized in ontologies and thesauri. A typical language for expressing annotations in the context of the Semantic Web is the Resource Description Framework (RDF, [20]) while the Web Ontology Language OWL ([19]) may be used to express semantic relationships between the concepts and resources used for annotation. There exist myriads of ontologies and parameter definitions for different application domains such as the Dublin Core parameters [3]) for editorial content.

Semantic annotation in current content management systems is usually restricted to preselected ontologies and parameter sets. Rich conceptual data models are only available in more sophisticated systems. Because most generic CMS are focused on Web content management semantic annotation is usually restricted to editorial parameters. Specialized content management systems which are adapted to
certain application domains incorporate preselected and tailored ontologies. Especially for XML-based content there exist several annotation platforms which incorporate semantical annotation either manually or semi-automatically; see [11] for a survey on available platforms.

Automated processing of semantical metadata is usually restricted to search facilities, e.g., searching for the author of an article. Because ontologies are preselected for most systems a full-featured reasoning support is usually not available. Especially for OWL ontologies there are reasoning tools based on description logics such as Racer ([7]) or FaCT which enable T-box (but also A-box) reasoning about semantic relationships between annotation concepts.

Applying generic semantical annotation and classical reasoning facilities to content management suffers from several drawbacks:

- Content as aggregated macro data is only partially analysable. The purpose of metadata is the description of properties which cannot be concluded from the data itself. The very simple annotation frame of \((S, P, O)\) triples does not allow one to express complex properties. For that reason this information has to be kept in the underlying ontology by defining appropriate concepts. The support of user-specific concepts increases the size of the ontology significantly and makes reasoning support even harder. Ad hoc definitions of user-specific concepts is not supported in this annotation model.

- Annotation with respect to arbitrary ontologies implies general purpose reasoning support by the system. Reasoning for even simple languages suffers from its high computational complexity (e.g., NEXPTIME for the restricted OWL-DL dialect, [9].) Dealing with high worst-case complexities implies a small size of input data but this is a contradiction to expressible ontologies and the definition of content as complex structured macro data. Especially the size of content instances is a crucial factor because A-box reasoning is a critical point for automated content processing ([8]).

But there are advantages, too:

- Usually, it is possible to distinguish between different points of view on content instances. Not every property is important while looking from every point of view. The macro data may encapsulate and hide properties from its aggregated micro data. Reasoning about the properties of the compound can be separated from the properties of the elements as well as the properties of interconnections between content instances.

- Typical application scenarios determine important properties and suggest evaluation strategies. So ontologies may be decomposed to enable a contextualized reasoning, e.g., on the basis of Local Model Semantics ([6]). Local reasoning may rely on a language that is just as expressive as needed in this context. Contexts relying on less expressive languages may support automated reasoning while contexts relying on more expressive languages may be used for manually interpreted information. Soundness and completeness of the reasoning process are not of primary interest as long as the reasoning result is acceptable in the application domain.

- The separation between annotations relying on common knowledge, user-specific annotations and (especially) usage-specific annotations reduces the size of incorporated ontologies significantly.

- If semantic annotations themselves are given a more sophisticated internal structure reasoning can be adapted to the requirements of the application domain.
The major disadvantage of current semantic description in content management is the treatment of knowledge over content instances as metadata on a secondary level in a strongly restricted language. In the following sections we will introduce a data model for content which handles the semantic part on the same level as the content itself and gives additional structure to the semantic description. We will start with the definition of content chunks as semantically enriched content instances in Section 2. In Section 4 we will introduce the notion of a schema for content chunks to incorporate typical functionality of content management systems such as content generation, content delivery, or content exchange.

As an example for a content management system we will take a look at a (simplified) Web shop application which sells products to customers via a website. The usual functionality should be supported: customers are able to search and browse for products, manage their profiles, shopping carts, and wish lists and order products.

2. Content Chunks

If we consider the HERM ([14]) schema fragment in Figure 1 we see a modelled list of products.

This modelling reveals certain structural properties such as attributes of the entity types and the connection between the list and the products. But the purpose of this model is missing. What kind of list was modelled: a shopping cart, a wish list, a list of stock items? Why was it modelled? What does the modeller expect? All this information is missing in the modelled fragment but is crucial if content instances of this schema are processed: if the list represents a shopping cart, pricing information may be collected. If it represents a wish list, there may be the need for additional functionality for discovering related products. It is obvious that expressing all this information by \((S, P, O)\) annotations will increase greatly complexity of each content instance and prevents a sophisticated semantic handling.

Modelling the semantics behind the data needs as much attention as modelling the structural properties of content. For that reason we propose a content data model which integrates structure, intension, usage, and origin on the same level. We start with the definition of content instances in this model.

**Definition 1.** A content chunk is a tuple \(C = (D, I, T, O)\) where \(D\) is the structural representation of a content instance, \(I\) an intensional description, \(T\) a topic description expressing the usage of content, and \(O\) the specification of the context where the content instance is used. The state of a content system is a finite set of content chunks.

Figure 2 visualizes the four dimensions of content chunks. Each content chunk expresses the association identifying ‘what (structure) is used by whom (origin, context) in which way (topic) under which assumptions and thoughts (intension)’. In the following paragraphs we will refine these notions.
The structural part of a content chunk reflects the classical notion of a content instance. Depending on the nature of the content data may be represented using an instance of a database schema formulated in ERM or UML, a semistructured resource such as a XML document, or a subsymbolic resource such as a picture.

**Definition 2.** Let \( \mathcal{L} \) be a set of (supported) modelling languages, \( S_\mathcal{L} \) the set of all schemas expressible with a certain modelling language \( L \in \mathcal{L} \) and \( \Sigma_S \) the set of all possible states of a certain schema \( S \). The structural component \( D \) of a content chunk \( C \) is a triple \((L, S, I)\) denoting a modelling language \( L \in \mathcal{L} \), a schema \( S \in S_\mathcal{L} \), and an instance \( I \in \Sigma_S \).

In our example, \( \langle \text{HERM}', s, i \rangle \) is the structural part of a content chunk if \( s \) denotes the schema in Figure 1 and \( i \) an instance which associates e.g., the entity type \textit{List} with the entity set \{\{\text{No} : 1\}\}, the entity type \textit{Product} with the set \{\{\text{No} : 134, \text{Descr} : \text{Book}, \text{Price} : 16.99\}, \{\text{No} : 521, \text{Descr} : \text{CD}, \text{Price} : 9.95\}\}, and the relationship type \textit{consistsOf} with the relationship set \{\{\text{List}.\text{No} : 1, \text{Product}.\text{No} : 134, \text{pos} : 1\}, \{\text{List}.\text{No} : 1, \text{Product}.\text{No} : 521, \text{pos} : 2\}\}.

The topic part of a content chunk is the conceptual counterpart to the presentation facilities of content management systems. Available systems offer template mechanisms (e.g., based on XSLT or scripting languages such as PHP or JSP) which transform a content instance to a physical representation ready for delivery through an output channel, e.g., HTML Web pages, e-mails, or PDF documents. Instead of coding presentation on the level of rendering templates a more abstract approach should be used. Topic maps ([10, 17]) provide the general data structure for a user-dependent view on content on the conceptual level. Expressing content via topic maps fulfills the following tasks during content delivery:

- The data structure is transformed to local vocabulary, e.g., according to a corporate identity or internationalization. In our example attribute names may be changed to language dependent labels. The prices of our products may be converted to local currencies or may be recalculated according to different tax regulations.

- The content is embedded into the usage context. The onion approach for a stepwise generation of delivered content ([15, 16]) defines different kinds of embeddings depending on the profile of a user (characterization of the properties of the user such as language, skill level, or preferences) and portfolio (tasks which have to be, should be, and can be fulfilled by the user, see [5].) This
information is obtained from the specifications of workflows and storyboards for interaction and added to the topic map as supplementary content. There are different kinds of supplementary content:

- **static content**, e.g., the logo of the company or statically linked elements such as advertisement banners,
- **decorative content** which is dependent on the current usage context but has no meaning to the application such as contextual help or integrated services such as a contextual weather forecast,
- **additionally delivered content** such as information about manufactures or links to related products in our Web shop example, and
- **navigational events** such as navigational links allow the user to interact with the system.

Multiple topic maps may be merged for multi-modal applications.

Supplementary content is incorporated in the topic map by parameterized queries on the set of content chunks and user specifications which characterize the occurrences of topics defined in the topic map. These queries are evaluated during content delivery and produce the topic map which can finally be rendered.

The topic part of a content chunk in our example may be the topic map depicted in Figure 3. This topic map reflects our product list in the context which supplies additional information on these products. This topic map can be transformed to a physical representation (e.g., a HTML page) using the usual techniques mentioned above.

To express content syndication information about the origin of content has to be stored. The provenance of data was already studied on the instance level ([2, 22, 1]) especially for scientific data sets. We can adapt these results for our purposes. We choose a finite set \( C \) from a universe \( \mathcal{U}_C \) of contexts. Each
context in $C$ represents a point of view on the application area under consideration. These points of view may be different points of view of the same user or may belong to different users. Because all these contexts are views on the same universe of discourse they are related: data, intensions, and topics may be exchanged between contexts. Actions in one context may affect other contexts.

**Definition 3.** Let $U_C$ be a universe of contexts and let $C \subset U_C$ be a finite set of contexts. Further, let $A = \{A_1, ..., A_n\}$ be a set of content chunks. The origin part of a content chunk $C$ is a tuple $(c, A)$ with a context $c \in C$ where the content chunk $C$ resides and a set $A$ of content chunks which are considered to be the ancestors of this chunk. The graph implied by this ancestor relationship between content chunks has to be acyclic.

Connections between content chunks enable the exchange and transformation of data, intensions, and topics between different contexts. In our example we may define a content chunk representing our product list together with a topic map for rendering a shopping cart. By adapting the topic map as well as the intension we may construct a content chunk which renders an order confirmation.

The intension of a content chunk expresses the purpose of the content as well as meanings and thoughts about the content chunk. Thoughts about some object may be expressed using a general description frame. A sophisticated and generic frame was given by Zachman in the context of specifications in software engineering ([23, 13]): each thought is expressed by formulating the facets who, what, when, where, how, and why. Each facet is specified by a concept. A concept itself is a small logical theory. We base our notion of concepts on intensional logics, especially on a restricted version of Transparent Intensional Logic (TIL) introduced by Tichý ([4]). TIL introduces the notions of intensional functions which map modalities (time, place, object identity, possibility, etc.) to values and intensional constructions building the intension of more complex expressions out of its components.

In our example we may introduce the concepts of customers, products, shopping carts, wish lists, product orders, etc. The concept shopping cart implies an intension of what a shopping cart is: it is a list of products selected from the offers in our Web shop. These products may be purchased in the future.

TIL analyses the intension of a concept down to the objectual base (calculating valuations of the intension behind the sentence ‘Products are associated with a description and a price’ considers all possible valuations of product, description, price, associated with and even and in ‘one shot’.) This is not the natural way of thinking. We modify the TIL approach in the following way:

- We introduce different types of individuals in the objectual base. TIL defines a single class $\iota$ of individuals. Introducing multiple (disjunct) classes $\iota_1, ..., \iota_n$ together with operations and predicates (such as ordering relations) corresponds to the definition of data types for attributes in classical database modelling. As defined in TIL there is at least one class $o$ of truth values ($true, false$) with the usual operations and predicates. The intension behind these operations and predicates is no longer transparent in terms of TIL.

- We support different types of modalities. TIL is specialized on modalities object identity and time and defines each intensional function on these modalities. Because there are other modalities (especially the possibility of a fact) and some intensions may be expressed in a more compact way if e.g., the time modality is omitted we will define intensional functions over arbitrary modalities from a given universe $\Omega$ of modalities.
• The objectual base consists of all first order types defined in TIL:
  - \( \iota, \omega \) and \( \omega \) are first order types,
  - each partial function \( \alpha_1 \times \alpha_k \rightarrow \beta \) with first order types \( \alpha_i \) and \( \beta \) is a first order type,
  - nothing else is a first order type.

**Definition 4.** An intensional function is a function \( f : \omega_1 \times \omega_k \rightarrow \alpha \) mapping possible worlds \( (w_1, ..., w_k) \) to instances of a first order type \( \alpha \). An intensional function is called non-trivial if there are two possible worlds \( (w_1, ..., w_k), (v_1, ..., v_k) \) with \( f(w_1, ..., w_k) \neq f(v_1, ..., v_k) \).

All first order types which are no intensional functions are called extensions.

Intensional functions can be used to express the usual type constructors: classes can be represented by their characteristic function, attributes by functions mapping to individuals, associations between objects by functions mapping to object identities.

In contrast to TIL we consider different kinds of intensional functions. The first kind is defined in a non-transparent way. Typical examples are extensional objects such as operations and predicates on the objectual base. Other non-transparent functions may be obtained from external data sources. For example, the concept of a customer may be represented as a characteristic function over modalities \( \omega \) (object identity) and \( \tau \) (time): \( isCustomer : \omega \times \tau \rightarrow o \). The valuation of this function may be determined by coupling the function with the customer database of our Web shop: \( isCustomer(w, t) = true \) if and only if the object with identifier \( w \) is registered as a customer in our customer database at time \( t \). Non-transparent intensional functions may be evaluated but do not reveal the internal structure of the valuation or their relationship to other intensional functions.

The second kind of intensional function is built in a transparent way: an intensional construction is used to relate valuations of the function with valuations of other first order types. Tichý introduced four types of constructions: variables of type \( \alpha \), trivialization (using objects in constructions), composition (application of values to a function) and closure (creation of functions).

**Definition 5.** We consider a single context \( c \in \mathcal{C} \). We organize intensional functions on strata in the following way:

- Operations and predicates on the objectual base (such as boolean connectives) as well as all non-transparent intensional functions and all intensional functions imported from contexts other than \( c \) are considered to be functions on stratum 0.

- Let \( k \) be an intensional construction with free variables \( x_i \) and a total mapping \( p : \mathcal{X} \rightarrow \mathcal{F} \) from variables \( \mathcal{X} = \{x_1, ..., x_n\} \) to intensional functions \( \mathcal{F} = \{f_1, ..., f_m\} \) where the stratum of \( f_j \) is at most \( s - 1 \). The intensional function constructed by \( k \) is considered to be a function on stratum \( s \).

The layering of intensional functions implies the independence of intensional functions on lower strata from intensional functions on higher strata and especially from their usage in constructions. This enables the determination of valuations of intensional functions on higher strata by first fixing the valuations of intensional functions on lower strata. This restricts expressiveness with respect to TIL. The strict monoton layering may be relaxed to constructions out of functions from the same stratum. Functions can be lifted to higher strata by using identity constructions, so we will allow the direct assignment of functions to strata higher than given by the definition.
Intensional constructions represent the terminological knowledge in traditional ontologies. Constructors such as ‘is-a’, ‘part-of’, or ‘union-of’ represent a fixed, preselected, and not configurable set of intensional constructions.

Building intensions by intensional constructions does not associate valuations of this intensional function with concrete objects. Beside intensional (T-box) reasoning based on constructions, properties of valuations of intensional functions have to be revealed.

Definition 6. Let \( c \) be a context, \( \mathcal{F} = \{f_1, ..., f_n\} \) a set of intensional functions, \( \mathcal{L} \) a logical language and \( \mathcal{T} \) a theory with sentences from \( \mathcal{L} \) formulating the knowledge about the valuations of \( \mathcal{F} \) with respect to the layering of intensional functions in \( c \). The tuple \( \mathcal{B} = (\mathcal{F}, \mathcal{T}) \) is called a concept in context \( c \).

In our Web shop example we might consider intensional functions \( isCustomer : \omega \times \tau \rightarrow o \), defined in a non-transparent way as mentioned above. Assuming that a customer will remain to be a customer for all the time we can express this in our small theory about customers:

\[
isCustomer(w, t) \Rightarrow (\forall t' > t)(isCustomer(w, t'))
\]

In another example shopping carts \((isShoppingCart : \omega \times \tau \rightarrow o)\) might become an order list \((isOrderList : \omega \times \tau \times \eta \rightarrow o \text{ for possibilities } \eta)\):

\[
isShoppingCart(w, t) \Rightarrow (\exists n' \in \eta, t' \in \tau)(isOrderList(w, t', n'))
\]

With the definition of concepts we can finally construct content intensions:

Definition 7. Let \( \mathcal{S} \) be a set of facets (e.g., according to the Zachman framework). A content intension is a set of functions \( i : \mathcal{S} \rightarrow \mathcal{B} \) mapping facets from \( \mathcal{S} \) to concepts from \( \mathcal{B} = \{\mathcal{B}_1, ..., \mathcal{B}_n\} \) in the current context.

3. Query Facilities for Content Chunks

The definition of content chunks as combinations of data, intension, topic, and origin enables several kinds of query facilities in content management systems. In the rest of the paper we use \( \mathcal{D} \) for the set of all structure definitions in the state of the CMS, \( \mathcal{T} \) for the set of all defined intensions, \( \mathcal{I} \) the set of all defined topic maps, and \( \mathcal{C} \) for the set of all contexts. Typical examples for query functions are:

- Structural queries remain unchanged. Depending on the modelling language(s) the usual query facilities are present, e.g., return all products from a certain product list.
- The function \( explain : \mathcal{D} \rightarrow 2^\mathcal{I} \) returns all intensions associated with a certain data instance.
- \( sample : \mathcal{I} \rightarrow 2^\mathcal{D} \) returns all data instances associated with a certain intension.
- \( express : \mathcal{D} \times \mathcal{I} \times \mathcal{C} \rightarrow 2^\mathcal{T} \) returns all topic maps associated with the given data object under the given intension in the given context.
- \( understand : \mathcal{T} \times \mathcal{C} \rightarrow 2^{\mathcal{I} \times \mathcal{D}} \) returns data instances together with an intension for the given topic map and context.
- \( find : \mathcal{C} \rightarrow 2^\mathcal{C} \) returns the contexts which incorporated content from this context.
• T-box reasoning in a generalized fashion is available by evaluating the intensional constructions. There is additional reasoning support, as depicted in Figure 4.

Properties of a concept relevant within a context are expressed in small local theories. We do not assume that this theory is a complete description of the concept but reveals relevant aspects. Concepts may be imported by other contexts while possibly different properties may become important. This is expressed by associating a different theory to the corresponding intensional function. For example, in a certain context it may be important to have a conception about the time when a person became a customer. An additional intensional function $customerRegistrationDate : \omega \rightarrow \tau$ may be introduced on a stratum lower than $isCustomer$ while the local theory of the concept $customer$ is enhanced by the constraint

$$(\forall w \in \omega)(\forall t < customerRegistrationDate(w)) (isCustomer(w, t) = false)$$

Evaluation of properties follows this construction strategy:

- First, the theory locally defined within the current context is used to prove the desired property.
- If the local proof was not successful, the intensional construction is investigated and reasoning is delegated to original contexts where the concept was imported from.

It is obvious that reasoning in this fashion does not ensure decidability but enables the delivery of precalculated relevant aspects which may not be accessible by pure intensional reasoning.

4. Content Schemas

In Section 2 we defined the building blocks of content as arbitrary tuples $(D, I, T, O)$. Considering typical application scenarios of content management systems arbitrary associations can be restricted to
support additional content management functionality:

- There are relationships between intensions and structural properties. Reasoning about intensional properties is reflected by certain values of the associated data instances. For example, reasoning about prices should be reflected by appropriate attributes in the structural definition. Non-transparently defined intensional functions must be directly computed from data.
- Information expressed in a topic map should be related to the underlying data and vice versa.
- Information can only be expressed or understood if there is an appropriate intension. On the other side, every intension should be expressible.
- Content which is imported from different contexts may not be completely revised but transformed.
- Not every intensional construction should be allowed. To restrict complexity a configurable set of construction templates has to be defined which incorporates the conceptual theories from the sources to build theories in the target context.

Restrictions may be expressed by constraint relations between the four dimensions of content chunks. To support content management functionality a mapping approach is better. There are three general tasks which have to be fulfilled during content management: content is created, selected content is delivered to the user, and content is exchanged between different contexts. Content creation in terms of our data model is the mapping of a topic map within a context to combinations of an intension and a data instance. Content delivery is the mapping between a data instance and an intension within a context to a topic map. Content translation maps content chunks from one context to another.

**Definition 8.** A content schema is a tuple \((\text{generate}, \text{deliver}, \text{exchange})\) with a function \(\text{generate} : \mathcal{T} \times \mathcal{C} \rightarrow 2^{\mathcal{D} \times \mathcal{T}}\), a function \(\text{deliver} : \mathcal{I} \times \mathcal{D} \times \mathcal{C} \rightarrow \mathcal{T}\), and a function \(\text{exchange} : \mathcal{D} \times \mathcal{I} \times \mathcal{T} \times \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{D} \times \mathcal{I} \times \mathcal{T} \times \mathcal{C}\).

These functions are defined for each context separately. First, a set of base intensions is defined. These base intensions rely on concepts (such as *customer* or *shopping cart*) which may be defined transparently or non-transparently. These base intensions are associated with a data schema \((L, S)\) (where \(L\) is a modelling language and \(S\) is a schema expressed in this language), a topic map template incorporating the data by associated data queries and a data template defining the data instance by queries on the topic map.

**Definition 9.** Let \(\{k_1, \ldots, k_n\}\) be a set of intensional constructions. An intensional construction template is a tuple \((\{k_1, \ldots, k_n\}, p, m)\) with intensional constructions \(k_i\) for each facet in the intension specification frame, a parameter assignment specification \(p : \mathcal{X} \rightarrow \mathcal{B}\) mapping variables from \(\mathcal{B} = \{x_1, \ldots, x_k\}\) to concepts from \(\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_l\}\) restricting valuations for variable substitutions in \(\{k_i\}\) to the given concepts, and a merging function \(m\) which creates

- the logical theory \(\mathcal{T}\) out of the theories associated to \(\mathcal{X}\),
- the data schema out of the data schemas of \(\mathcal{X}\),
- the topic map template out of the topic map templates of \(\mathcal{X}\), and
- the data template out of the data templates of \(\mathcal{X}\).

The definition of the data schema, the topic map template, and the data template implies the content generation and content delivery functions. The creation of the logical theory out of other concepts is given by a compatibility relation between models of these theories as defined by the Local Model Semantics framework ([6]).
5. Semantic Wikis: Enabling Collaborative Content Annotation and Foundation

Communities form an interacting group of various actors in a common location, common intension, and common time. They are based on shared experience, interest, or conviction, and voluntary interaction among members with the intension of members welfare and collective welfare. They can have more or less structure and more or less committed members.

Wikis are the equivalent of content management systems in such communities. Community members are allowed to instantly change the content (usually Web pages.) We can extend this conception of Wikis and look forward on how functionality of Wikis may evolve by incorporating topically annotated and intensionally founded content.

Content creation and content annotation are resource-intensive processes. Introducing a user- and usage-centric approach to content handling as presented in this paper, these processes can be distributed through a social network, adapting the notions of the Web 2.0 initiative.

Wikis are considered to be special content management systems which allow the user to instantly change the content (usually Web pages.) We can extend this notion to semantically enriched content:

- Content may be loaded into the system. This reflects the usual process of editing pages in a Wiki. The load process results in stored data instances in the CMS which can be extracted via search templates or associated with metadata in the usual sense (e.g., editorial parameters).

- Data instances may be associated with intensional descriptions such as copyrights and access rights.

- The user may annotate the content after searching for it, e.g., making recommendations on products in our Web shop example. A recommendation can be expressed by an additional intension on the content chunk expressing that the current user interprets the data as a product recommendation. The local theory expresses the fact, that this user has bought these products or might buy these products in the future.

- Another user may explore the notion of a ‘recommendation’ from the context of the first user if he sees the same data instance and looks for associated intensions. Afterwards, this user may use this concept to annotate other data instances.

- Users may refine the local theory of a concept to incorporate knowledge which was hidden so far.

- Users may associate new topic maps to content to create different (e.g., localized) versions.

Beside supporting content generation and annotation by social networking, semantically and user-specifically enriched content chunks are the base for modelling collaboration within a network of users. Collaboration is seen ([21]) as a process of interactive knowledge exchange by several people working together towards a common goal. Collaboration can be characterized ([15]) by three facets: communication, cooperation, and coordination. The communication facet defines the exchange protocols of content.
between users. The cooperation facet relies on the workflow of the collaboration by specifying who (actor) has to deliver which results to whom. The coordination facet defines the task management and synchronization between the collaborating partners to fulfill the cooperation goals.

Collaboration in social networks is usually defined in an implicit and decentralized way, so classical workflow management systems with fixed process definitions cannot be applied. The content data model defined in this paper can be used to annotate content with the specified collaboration frame to express

- the history of content and content changes,
- the purposes of content changes and content usage,
- future tasks on content chunks and therefore
- access rights on content chunks.

In the context of collaboration the specification of users becomes important. Conceptually, users are handled by a set of concepts \( A \) called actors (such as administrator, moderator, registered user, unexperienced user, guest user, etc.) Actors define roles of users and therefore imply a grouping on the set of users. According to our definition of concepts each actor is associated with a small logical theory expressing the properties which are common to all users in the user group of the actor.

Communication between users takes place by topics. Topic map fragments have to be defined in the content schema to express tasks in the collaboration frame characterized above. Figure 5 shows an example for a topic map concerning a product within our Web shop. A typical task which can be done by users is to write a comment. The topic map for expressing the product incorporates only comments fulfilling the intension of a proofread comment. To write a comment the topic map is merged with a topic map requesting comments from users. Occurrences of these topics are linked with dialog specifications that offer access to the CMS services to fulfill the desired task.

These topic map fragments which express task specifications are associated with an intension according to our collaboration frame (who wrote a comment on which product.) in the content schema. Coordination is done by expressing obligations (e.g., adoptions of [18]) on the content chunk in the local theory of the intension, e.g., a moderator has to proofread the comment after the comment was written.
and before the comment is published. For that reason there is a topic map defining the proofreading task for moderators which can be merged with the topic map associated with the intension of commented products. This merging process creates the intension of a proofread comment, characterized by the fact in the local theory that at a time point \( t \) the proofreading task took place:

\[
isProofReadComment(w) := (\exists w', t < now)(moderator(w, t) \land proofread(w, w'))
\]

The new query functionality on content with topic annotations and intensional foundations enables additional high-level functionality for content management systems in social environments to support more sophisticated content management services:

- Sophisticated and automatically generated graphical user interfaces such as WYSIWYG editors rely on a user-centric topic-based content annotation to provide information in the right fashion at the right time.
- Community services such as contributions to communities, joining communities, meetings, publications, or community organization as well as their integration can be intensionally modelled.
- Online collaboration support active discussion and interaction among participants as well as content exchange.
- Content changes within a collaborating environment may be conflicting. Expressing the purposes of changes may help to solve these conflicts.
- Task annotations support modelling of interaction scenarios and coordination facilities such as schedules to enable project management functions.
- Secretarial functions such as filtering or ordering can be intensionally expressed and enforced by appropriate topic map definitions.
- Blackboard facilities support tracing of tasks, members, schedules, and documents. Report functions may be incorporated for non-members of the community.
- Ad hoc (and implicit) communities are supported. Members can conduct their own communities, interests, tasks, and portfolios by defining private workspaces.
- Asynchronous as well as synchronous collaboration is supported depending on the handling of annotations and intensions.

6. Conclusions

In this paper we are introducing a data model for content management systems which handles content as associations between the data itself, the intension behind the data, the usage of data and the origin of data. Content annotation is treated according to its purpose: terminology which is common sense in a community is shared in ontologies. Concepts which are only relevant to certain users or in certain situations are defined locally and removed from the global ontologies to make reasoning about global terminology easier. Local concepts may be exchanged and adapted between different usage contexts.
For that reason concepts are seen not only as notions from an ontology but as small logical theories. Additionally, intensional annotation is separated from usage annotation to allow different expressions of the same data under the same intension.

Because of the reduced size of annotation ontologies, the local definition of concepts and the suggested evaluation strategies according to the origin definitions of the content, the separation of concerns within the data model allows a better automated reasoning support than simple \((S, P, O)\) annotation frameworks although decidability as well as soundness and completeness of the reasoning process cannot be guaranteed. The user-centric approach together with the facility of explicitly incorporating and exchanging hidden knowledge into local theories behind a concept ensure the usability within known application scenarios when automated reasoning fails. Adding local and user-specific semantics to content chunks is a prerequisite for distributing content over social networks and therefore extends current Web 2.0 technologies in a natural way. While today Wikis support open communities mainly interested in free-form changes of content, Semantic Wikis may also support transaction oriented communities with the need of at least partially controlled collaboration.

References

ASM Foundations of Database Management

Bernhard Thalheim\(^1\) and Klaus-Dieter Schewe\(^2\)

\(^1\) Christian Albrechts University Kiel, Department of Computer Science
Olshausenstr. 40, D-24098 Kiel, Germany
\(^2\) Massey University, Information Science Research Centre
Private Bag 11 222, Palmerston North, New Zealand
thalheim@is.informatik.uni-kiel.de, k.d.schewe@massey.ac.nz

Abstract. Database structuring is well understood since decades. The operating of databases has been based in the past on temporal logics and did not yet get an easy to understand formal underpinning. Therefore, conceptions like transaction and recovery are mainly discussed at the logical or operational level. This paper shows that database structuring and functionality can be defined within a uniform language. We base database semantical on the operational semantics of abstract state machines (ASM). This uniform mechanism allows to define the structuring, the functionality, the distribution and the interactivity of a database system in a way that supports abstract consideration at various layers of abstraction, that supports refinement of specifications to more detailed ones and that support proof of properties.

1 Adequacy and Deficiencies of Database Technology

1.1 Strength and Weaknesses of Database Technology

Database systems are currently broadly used for support of data-intensive services. These broad usage is based on advantages such as the following:

Consistent storage of data: Data are uniquely stored in the most actual version. Each user gets the same data. Inconsistency can be avoided. Furthermore, redundancy can be reduced and standards can be enforced.

Multi-user support: Data can be consistently shared among different users. Also, conflicting requirements can be balanced. Security is enforced by restricting and managing access to data. Data can be consistently distributed within a network.

Integration into component-ware: Currently, database systems are turned into middle-ware components in information-intensive applications. Database operating is based on the transaction paradigm. A transaction is a logical unit of work. Database systems are designed to support transactions.

Nevertheless, database engines do not completely support complex applications such as internet services, real-time applications, stream information systems and web information systems.
**Content and information instead of data:** With the development of more complex application sophisticated support for various abstraction levels on data is required. Instead of raw or micro-data users want to retrieve condensed data or macro-data and to work on them.

**Transaction semantics:** The transaction approach is based on atomicity, consistency, isolation and durability requirements on transaction application. User action are sometimes not atomic and require sophisticated support for intermediate actions. The isolation level may vary over time. The are different degrees for durability.

**View support:** Each object in the database needs to be identifiable. Query languages allow to construct macro-data from the micro-data of the objects contained in the database. These macro-data may be composed into complex derived objects. Due to the query language some of them may be not identifiable. Further, their connection to the micro-data may be not injective. Therefore, data manipulation on macro-data cannot translated to data manipulation on micro-data.

**Missing operational semantics:** Semantical treatment of structuring is based on the predicate logic since the advent of the relational database model. Entity-relationship modelling can be based on a generalized predicate logic. Object-oriented models are often defined in a fuzzy manner without existing models. Moreover, database operation is still not well based.

In this section we show now that the weaknesses of database models and technology for internet application can be reduced by the ASM approach.

### 1.2 Well-Founded Structuring and Operating Transparency

Database systems are typically specified through a specification of database structuring and the assumption of canonical database operating. Structuring of databases is given by a signature of the database and a set of static integrity constraints. They form the schema of the database. The signature is used as an alphabet of a canonical first-order (or first-order hierarchical [13]) predicate logic. Static integrity constraints may be expressed as formulas of this logic. The functionality of a database system is defined on the basis of an algebra that is generically defined over the signature. This algebra is extended to a query algebra and to modification operations for the database system. Database management systems provide services that are assumed to be given whenever we are talking on database operating. Their behaviour and their operating is assumed to be canonically given.

This assumption might be appropriate for the predesign or business user layer [5,13] of specification. The implementation independence of the business user layer and the operating transparency supports concentration application specific aspects of a database system. It is not appropriate for the conceptual layer since we need to consider database behaviour as well. This inappropriateness causes many problems for database programming and operating. This paper shows how the ASM approach can be used to overcome this gap.
Information systems extend database systems by a support for users, by interface systems and by content management. Web information systems [9,12] augment classical information systems by modern web technologies. They aim in supporting a wide variety of users with a large diversity of utilisation 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 [10] has already partially been supported.

1.3 Necessities for Specification of Interactive Information Systems

Despite the presence of an active research community that studies conceptual models for information systems, interaction still lacks precise formal underpinnings. There are several approaches which can be generalized for development of a formal basis of interactive information services:

**Workflow and pattern:** Workflow approaches allow to combine dataflow and computation. Workflows can be constructed on the basis of basic processes by application of basic control constructors such as sequence, parallel split, exclusive choice, synchronization, and simple merge, by application of advanced branching and synchronization control constructors such as multiple choice, multiple merge, discriminator, n-out-of-m join, and synchronizing join and by application of structural control and state-based control constructors.

**Wegner’s interaction machines:** The approach is based on four assumptions: User rather observe machine behavior instead of having a complete understanding of machines I/O behavior. Modelling of systems cannot be entirely inductive. Instead of that co-induction needs to be used, at least for the earlier design phases. Users compete for resources. Thus, modelling includes an explicit specification of cooperation and competition. Interactive behavior cannot be entirely modelled on the basis of I/O behavior but rather in term of interaction streams.

We use the interaction machine approach [6] in order to reason formally on information services. An interaction machine [15] can be understood as Turing machine with multi-user dynamic oracles (MIM) or with single-user dynamic oracles (SIM). An interaction machine can be specified as follows [3]:

```plaintext
if condition
    then state = Nextstate(state, database, input)
        database = Modification(state, database, input)
        output = output ◦ Out(state, database, input)
```

**Statechart diagrams:** The statechart approach has been proved to be useful for state-oriented modelling of database applications and more generally for use-case diagrams. Despite its usage in UML it has a precise semantics which can be easily integrated into ER modelling approaches [13].
User interaction modelling: Interaction involves several partners (grouped according to characteristics; group representatives are called ‘actors’), manifests itself in diverse activities and creates an interplay between these activities. In order to develop usable websites the story of the application has to be neatly supported [9,12,10,14]. Interaction modelling include modelling of environments, tasks and actors beside modelling of interaction flow, interaction content and interaction form [7].

1.4 Achievements of the ASM Approach

The abstract state machine (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 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 [16] of computer science: structuring, evolution, collaboration, and abstraction.

![Fig. 1. The four principles of Computer Science](image)

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.

2 ASM Specification of Database Systems

Database systems are designed to operate in parallel. Operating of databases systems can be understood in a state-based approach.

2.1 ASM Specification of Databases and Database Structuring

The abstract state machine approach allows simple and refinable specification of parallel processes based on states and transitions.

The ASM signature $\mathcal{S}$ is a finite collection of function names.

- Each function name $f$ has an arity, a non-negative integer.
- Nullary function names are called constants.
- Function names can be static or dynamic.
- Every ASM signature contains the static constants $\text{undef}$, $\text{true}$, $\text{false}$.

A database schema may be based on a collection of (predicative) functions representing the structures of the database. Typically such functions are dynamic. Static functions are those functions that do not change over time. Most generic database computation functions such as the aggregation functions can considered to be static. They are defined as static higher-order functions. Their concretisation to database functions may lead to a dynamic functions or may still be static. Database state functions are however dynamic.

The signature $\mathcal{S}$ is the main component of alphabet for the logical language $\mathcal{L}_\mathcal{S}$. We assume that this language is specified in the canonical way used in predicate logics. An ASM database schema $\mathcal{S}$ is given by a signature $\mathcal{S}$ and by a finite set $\Sigma$ of formulas from $\mathcal{L}_\mathcal{S}$.

We restrict the consideration in this paper to the tuple (or product) constructor. The set, list, multiset, disjoint union, labelling and naming constructors can be treated in a similar way.

A database state $\mathcal{DB}$ (or database instance) for the signature $\mathcal{S}$ is a non-empty set $Val$ called the superuniverse of $\mathcal{DB}$, together with an interpretation $f^{\mathcal{DB}}$ of each function name $f$ in $\mathcal{S}$.

- If $f$ is an n-ary function name of $\mathcal{S}$, then $f^{\mathcal{DB}} : Val^n \rightarrow Val$.
- If $c$ is a constant of $\mathcal{S}$, then $c^{\mathcal{DB}} \in Val$.
- The superuniverse $Val$ of the state $\mathcal{DB}$ is denoted by $Val(\mathcal{DB})$. 
Relations are functions that have the value true, false, undef

\[(\bar{a} \in R \iff R(\bar{a}) = true).\]

\[\text{dom}(f^{DB}) = \{(a_1, \ldots, a_n) \in Val(DB)^n \mid f^{DB}(a_1, \ldots, a_n) \neq \text{undef}\}\]

\[\text{rel}(f^{DB}) = \{(a_1, \ldots, a_n) \in Val(DB)^n \mid f^{DB}(a_1, \ldots, a_n) = \text{true}\}\]

The superuniverse can be divided into subuniverses represented by unary relations. These unary relations form the basic data types under consideration. A data type is typically given by a set of values and a set of (static) functions defined over these values.

A database state \(DB\) is a model of \(\Sigma\) (or is called consistent database state) if \([\Sigma]^{DB}_\zeta = \text{true}\) for all variable assignments \(\zeta\) for \(\Sigma\). We typically consider only models. Since transactions may have intermediate database states that are not models we distinguish the two notions. The distinction between undef and false allows to separate for relational functions the case that an object does not belong to the database from the case that an object is known to be false.

2.2 Specification of Database Operating

Database state modifications can be described as changes to the dynamic functions. Changes we need to consider are either assigning a value undef, true, false to a relational function or changes of the functions themselves. We thus may consider that a change can be given through a set of changes to the functions for the values of the domain of a function. This detailed view on the content of the database allows the introduction of database dynamics. Roughly speaking, we introduce memory cell abstractions (called locations) and consider the database to consist of those objects which locations evaluate to true. The functions of the database system can also be treated on the basis of locations.

A database location of \(DB\) is a pair

\[l = (f, (a_1, \ldots, a_n)) \quad ((\text{relational}) \text{ function, object}).\]

The value \(DB(l) = f^{DB}(a_1, \ldots, a_n)\) is the content of the location \(l\) in \(DB\).

The tuple \((a_1, \ldots, a_n)\) represents a potential object. The current value is given by a location \(DB(l)\) of the database system. The database consists of those objects \((a_1, \ldots, a_n)\) for which a relational function \(f(a_1, \ldots, a_n)\) evaluates to true.

An modification \((l, u)\) to \(DB\) is the assignment of a value \(u\) to a location \(l\). For instance, an insert changes the value \(DB(l)\) at a location \(l\) to true. The delete operation changes the value at a location \(l\) to undef. A basic database update assigns \(v\) to another location and changes the value of original location to undef. A modification is called trivial if \(v = DB(l)\). A modification set consists of a set of modifications.

A modification set \(U\) is consistent, if it has no clashing modifications, i.e., if for any location \(l\) and all elements \(v, w\) if \((l, v), (l, w) \in U\) then \(v = w\).

The result of firing a consistent modification set \(U\) is a new database state \(DB + U\)

\[(DB + U)(l) = \begin{cases} v & \text{if } (l, v) \in U \\ DB(l) & \text{if there is no } v \text{ with } (l, v) \in U \end{cases}\]

for all \(l\) of \(DB\).
For value-based types we define generic functions beyond \textit{insert}, \textit{delete}, \textit{update} and evaluation functions such as \textit{average}, \textit{min}, \textit{max}, \textit{sum}. Additional functions \((F_f)_{f \in F}\) are defined in the same manner on the type system. These functions can be used to construct a \(S\)-algebra (e.g., operations such as \textit{projection}, \textit{join}, \textit{selection}, \textit{union}, \textit{difference}, \textit{intersection}). We combine these functions into an \(S\)-algebra.

Databases change over time. We may represent the change history of a database by a sequence of database states. The changes are defined by an application of database systems operations to the current database state. We may require that any change applied to a model shall either transform the model to a new model or shall not be considered otherwise. This operating requirement leads to \textit{transaction semantics} of database operating. We can use the logic \(L_S\) for the definition of transition constraints.

A \textit{transition constraint} consists of a pair of formulas \((\psi_{\text{pre}}, \psi_{\text{post}})\) from \(L_S\). The transition constraint is valid for a database modification from \(DB\) to \(DB'\) if \(DB \models \psi_{\text{pre}}\) and \(DB' \models \psi_{\text{post}}\). Static integrity constraints from a finite set \(\Sigma\) can be mapped to transition constraints \((\land_{\alpha \in \Sigma} \alpha, \land_{\alpha \in \Sigma} \alpha)\). Let \(\Sigma_{\text{dynamic}}\) be the set of transition constraints.

A sequence of database states \(DB_0, DB_1, ..., DB_{i+1} = \tau_A(DB_i), ...\) satisfying \(\Sigma_{\text{dynamic}}\) is called a \textit{run} of the database system. The run can be defined though \(S\)-algorithms \(A\) or transformation rules that impose a one-step transformation \(\tau^D B\) of a database state to its successor in the run.

A \textit{database program} is given by a rule name \(r\) of arity \(n\) is an expression \(r(x_1, ..., x_n) = P\) where \(P\) is a database transition operation and the free variables of \(P\) are contained in the list \(x_1, ..., x_n\).

Database transition operations are either basic operations from the \(S\)-algebra or are constructed by inductively applying the following construction rules:

- Skip rule: \(\text{skip}\)
- Update rule: \(f(s_1, ..., s_n) := t\)
- Parallel execution rule: \(P\text{ par }Q\)
- Conditional rule: \(\text{if } \phi \text{ then } P \text{ else } Q\)
- Let rule: \(\text{let } x = t \text{ in } P\)
- For all rule: \(\forall x \text{ with } \phi \text{ do } P\)
- Choose rule: \(\forall x \text{ with } \phi \text{ do } P\)
- Sequence rule: \(P\text{ seq }Q\)
- Call rule: \(r(t_1, ..., t_n)\)

A database system consists of a database management system and of a number of databases. Let us consider only one database.

An \textit{abstract database system} \(\mathcal{M}\) consists of a

- a signature \(S_M\) of the database system that embodies the signature \(S\) of the database,
- a set of initial states for \(S_M\),
- a set of database programs,
- a distinguished program of arity zero called \textit{main program} of the system.

We denote the current state of the database system by \(DBS\) and by \(DB\) the current state of the database. The transition operation \(P\) yields the modification set \(U\) in a state \(DBS\) under the variable assignment \(\zeta: \text{yields}\)(\(P, DBS, \zeta, U\)).
Semantics of transition operations defined in a calculus by rules:

\[
\begin{array}{c}
\text{Premise}_1, \ldots, \text{Premise}_n \\
\hline
\text{Conclusion} \\
\text{Condition}
\end{array}
\]

A query is an (open) \( S \)-formula. Moreover, a view is a query adorned by names (labels) for free variables. Due to our definitions, views are hierarchical too.

### 2.3 ASM Specification of Database System Behaviour

The database system state \( \text{DBS} \) is structured into four state spaces:

\( (\text{input states } I, \text{ output states } O, \text{ DBMS states } E, \text{ database states } DB) \).

The input states \( I \) accommodate the database input to the database systems, i.e. queries and data. The output space \( O \) allows to model the output data and error messages. The internal state space \( E \) of the DBMS represents the DBMS states. The database content of the database system is represented in the database states \( DB \). The four state spaces are typically 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 can be structured accordingly.

The main database functions are modelled by modification, retrieval or internal control programs that are run in parallel:

- **Modification programs** allow to modify the database state if is enabled:

  \[
  \begin{align*}
  \text{if } I'(req) & \neq \lambda \land E'(\text{modify}) = \text{enabled} \land I'(req) \in \text{Update} \\
  \text{then } I'(req) & := \lambda, O'(\text{errMsg}) := \ldots, D' := \ldots, E' := \ldots
  \end{align*}
  \]

- **Retrieval rules** allow to retrieve the content of the database:

  \[
  \begin{align*}
  \text{if } I'(req) & \neq \lambda \land E'(\text{retrieve}) = \text{enabled} \land I'(req) \in \text{Update} \\
  \text{then } I'(req) & := \lambda, O'(\text{errMsg}) := \ldots, O'(\text{answer}) := \ldots
  \end{align*}
  \]

- **DBMS control rules** allow to change the database state and the internal state of the database:

  \[
  \begin{align*}
  \text{if } E'(\text{DBMSstateChange}) \land E'(\text{modify}) = \text{disabled} \\
  \text{then } D' := \ldots, E' := \ldots
  \end{align*}
  \]

The description can be extended in a similar fashion to sets of inputs instead of a singleton input.

The ASM programs are based on ASM rules:

- \( \text{MODIFYINPUT}(\text{request, DBMS\_state, DB\_state}) \),
- \( \text{MODIFYOUTPUT}(\text{request, DBMS\_state, DB\_state}) \),
- \( \text{MODIFYDB}(\text{request, DBMS\_state, DB\_state}) \),
- \( \text{MODIFYCONTROL}(\text{request, DBMS\_state, DB\_state}) \),
- \( \text{RETRIEVEOUTPUT}(\text{request, DBMS\_state, DB\_state}) \),
- \( \text{RETRIEVEDB}(\text{request, DBMS\_state, DB\_state}) \),
- \( \text{RETRIEVECONTROL}(\text{request, DBMS\_state, DB\_state}) \),
- \( \text{CONTROLLERDBMS}(\text{DBMS\_state, DB\_state}) \), and
- \( \text{CONTROLLERDB}(\text{DBMS\_state, DB\_state}) \).

These rules run in parallel. They express general DBMS functions for state modification:

- **modify**:

  \[
  (\text{req}, s, d) \implies (\_\text{errMsg}, s', d')
  \]

- **retrieve**:

  \[
  (\text{req}, s, d) \implies (\_\text{answer, errMsg}, s, d)
  \]

- **controller**:

  \[
  (s, d) \implies (\_\text{answ, errMsg}, s, d)
  \]
Modifications may cause deadlock. In order to overcome them the controller enables scheduling, recovery, and optimization. An modification can be imposed completely to the database for support of transaction semantics.

Summarizing we observe that ASM’s can be used for definition of operational semantics of database systems.

2.4 Co-design of Structuring, Interaction and Behavior

Information systems design starts with database structuring and bases functionality on structuring. Typically, it uses various abstraction layers: application domain layer for a rough specification of the application domain and its influence on the information system, requirements acquisition layer for the description of requirements such as main business data and business processes, business user layer or the predesign layer for the specification of business user objects and business workflows, conceptual layer for the conceptual description of structuring and functionality and implementation layer describing the implementation, i.e. code, SQL structures, interfaces and user views. Nowadays application tend to be distributed and components collaborate with each other. Information systems provide services to users depending on their application tasks, portfolio, user stories and context. Therefore, we represent the four dimensions of information systems specification and their layers in Figure 2.

![The abstraction layer model for information systems specification](Image)

Fig. 2. The abstraction layer model for information systems specification

The structuring and functionality can be described as an abstract state machine. The stories of users, their profile, context and portfolio can be combined into an ASM-based description of the story space. Interaction description can be based on notions developed in linguistics and in movie business.
According to the co-design approach we can distinguish two levels of consideration:

**Database system development layers:** The specification of database systems can be of different granularity and levels of detail. Typical development layers are motivation and requirements layers. A development layer (which is at the same time a database system run layer) is the conceptual layer. These layers display a database specification on different levels of detail and should be refinements of each other.

**Database systems operating layers:** Database operating is observed, maintained and administrated at various levels of detail: Business users have their own view onto the database application. These views may be different from each other. They are considered to be external views on the conceptual layer. The conceptual layer integrates all different aspects of database systems specification. The implementation layer is based on the logical and physical database schemata. Additionally, database systems functionality is added to the application on the basis of programs etc.

The ASM specification should match all different layers:

**Level 1 (Point of view of business users):** Database system defined by three state space: input state, database state, output state. The *input state* is based on algebraic structure with ground terms defined on the values and the names. The *database state* is based on the (object-)relational structure with well-defined composition operators. The *output state* is a general database defined on the values and names.

**Level 2 (Conceptual point of view):** Database systems are defined as an extension of level 1 by transactions, constraints, views and integrity maintenance.

**Level 3-1 (Logical point of view):** The logical database system defined as an extension of level 2 by states of the database management system by the transaction and recovery engine, by the synchronization engine, the logging engine and the query translating engine.

**Level 3-2 (Physical point of view):** The physical database system is defined as an extension of level 3-1 by specific functions of the DBMS.

**Level 4 (DBMS point of view):** On level 4, the storage engine is modelled in detail with the buffers and the access engine.

This separation allows to model database systems at different levels of detail. Each higher level should be a refinement of the lower levels.

### 3 Faithful Refinement of Database Systems Specification

The co-design approach can be based on the abstraction layer model. We need a correctness criterion for the faithfulness of specifications among these different layers. We may distinguish between specifications at the requirements acquisition layer, at the business user layer and at the conceptual layer. (Static) integrity constraints typically limit the state space for the instances of database schema.
Functionality can be provided within a framework for specifying the functions. The database system development process aims in stepwise refinement of structuring and functionality of a database system. This refinement process needs a formal underpinning. We can develop a theory of specification refinement for database applications based on ASM.

3.1 Refinements of Database Systems

Given two abstract database systems $M$ and $M^*$. The refinement of $M_{DB}$ to $M^*$ is based on

- a refinement of signatures $S_M$ to $S_{M^*}$ that associates functions and values of $S_M$ with those on $S_{M^*}$,
- a states of interest correspondence between the states of interest $DBS$ and $DBS^*$ defined over $S_M$ and $S_{M^*}$ correspondingly,
- abstract computation segments $DBS_1, ..., DBS_m$ on $M$ and $DBS^*_1, ..., DBS^*_n$ on $M$ and $M^*$,
- locations of interest $(DB, DB^*)$ defined on $S \times S^*$ and
- an equivalence relation $\equiv$ on locations of interest.

$M^*$ is a correct refinement of $M$ if there for each $M$-run $DBS^*_0, ..., DBS^*_k, ...$ there is an $M$-run and sequences $i_0 < i_1 < ...$ and $j_0 < j_1 < ...$ such that $i_0 = j_0 = 0$ and $DB^*_i \equiv DB^*_j$ for each $k$ and either
- both runs terminate and their final states are the last pair of equivalent states, or
- both runs and both sequences are infinite.

A refinement is called complete refinement if $M_{DB}$ is a correct refinement of $M^*_D$, and $M^*_{DB}$ is a correct refinement of $M_{DB}$.

We distinguish between
- structure refinement that is based on a notion of schema equivalence and state equivalence,
- functionality refinement that is based on a notion of schema equivalence and a notion of coherence, and
- state refinement that uses the notion of equivalence and coherence.

3.2 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:

**Inside-out refinement**: Inside-out refinement uses the current 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 are not yet considered.
**Bottom-up refinement:** Bottom-up refinement uses composition and generalisation of functions and of rules to more general or complex. Bottom-up refinement also uses generation of new functions and rules that are not yet 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 to derive plans for refinement and and *primitives* for refinement.

### 3.3 Generic Refinement Steps and Their Correctness

An engineering approach 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 [8] has observed that refinement steps can be governed by contracts. We may consider a number of governments [1]. We should however 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,

![Diagram](image)

**Fig. 3.** The derivation of correct refinement steps
submachine refinement, and \((m,n)\) refinement. The generic refinement is adapted to the assumptions made for the given application and to consistency conditions. Typical such consistency are binding conditions of rules to state and vocabulary through the scope of rules. The general approach is depicted in Figure 3.

4 Conclusion

The ASM approach allows the development of a theory of database system operating. It has the following advantages:

– The signature of object-relational databases can be specified based on hierarchical predicate logic. Therefore, the theory of static integrity constraints can be entirely embedded into the theory of sequential abstract-state machines [4].

– The run of a database system is parallel. As an example that is not covered in this paper we elaborated transaction semantics that handles conflicts in concurrency. Parallel runs of transactions must behave in the same manner as a sequential run of the transactions. This concept is entirely covered by partially ordered runs.

– Interaction of database systems with user or information systems can be handled by ASM as discussed above.

– Database systems are considered on a variety of abstractions layers. Users consider a database system as an input-output engine with a powerful memory. At the conceptual layer, a database system is far more complex. The implementation is even more complex. Therefore, the consideration of database systems semantics requires also a powerful theory of refinement as provided by ASM.

Our main aim in this paper is to develop a theory of database systems at the user layer. The other abstraction layers may be seen as refinements of the business user layer. We omitted all examples to the space limitations.

The next abstraction level must consider specific approaches of database engines at the implementation layer. In this case, we need a general notion of database semantics. Database dynamics has not yet been well understood. Transactions may serve as an example. So far we have only used sequential ASM. We might however use power ASM that allow to abstract from intermediate computational steps.

Acknowledgement

The authors would like to thank Egon Börger and to Andreas Prinz for their helpful comments and discussions. Egon Börger proposed to use power ASM instead of sequential ASM for the treatment of transactions. We than P. Schmidt for proofreading.
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 \( \oplus \) is attached to the relationship type that uses the cluster type. In this case directed arcs associate the \( \oplus \) 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 \( E \doteq (\text{attr}(E), \Sigma_E) \) 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 \( \text{def} \) of a type \( T \) is denoted by \( T \doteq \text{def} \).

The EER model lets users inductively build relationship types \( R \doteq (T_1, \ldots, T_n, \text{attr}(R), \Sigma_R) \) 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 \( C \doteq T_1 \cup \ldots \cup T_n \) 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 \( (..) \), set constructor \( \{..\} \), and the list constructor \( <..> \). Types may be optional component types and are denoted by \([..]\).

The types \( T \) can be labelled \( l: T \). 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:

\[
T = B \mid (l_1 : T, \ldots, l_n : T) \mid \{T\} \mid \langle T \rangle \mid [T] \mid T \cup T \mid l : T \mid N \doteq T
\]

**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 is the database itself representing a state of the application’s data often called micro-data. The fourth 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 \( (\ldots) \), set constructor \( \{ \ldots \} \), and the list constructor \( \langle \ldots \rangle \). 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. \( \text{dom}(A) = B \). 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 \( \text{dom}(A) \). Components of complex attributes may be optional, e.g., the \( \text{Title} \) in the attribute \( \text{Name} \).

Typical examples of complex and basic attributes in Figure 1 are

\[
\begin{align*}
\text{Name} & \triangleq (\text{FirstName} <\text{FirstName}>, \text{LastName}, [\text{AcadTitles}], [\text{FamilyTitle}]) \\
\text{PersNo} & \triangleq \text{EmplNo} \cup \text{SocSecNo} \\
\text{AcadTitles} & \triangleq [\text{AcadTitle}] \\
\text{Contact} & \triangleq (\text{Phone}([\text{PhoneAtWork}]) \cup \text{private}), \text{Email}, \text{URL}, \text{WebContact}, [\text{Fax}([\text{PhoneAtWork}])] ) \\
\text{PostalAddress} & \triangleq (\text{Zip}, \text{City}, \text{Street}, \text{HouseNumber})
\end{align*}
\]

for \( \text{DateOfBirth} :: \text{date}, \text{AcadTitle} :: \text{acadTitleType}, \text{FamilyTitle} :: \text{familyTitleAcronym}, \text{Zip} :: \text{string7}, \text{SocSecNo} :: \text{string9}, \text{EmplNo} :: \text{int}, \text{City} :: \text{varString}, \text{Street} :: \text{varString}, \text{HouseNumber} :: \text{smallInt} \).

The complex attribute \( \text{Name} \) 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 \( \text{ID}(K) \). 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 \( \text{id}(E) \) of \( \text{attr}(E) \), and a set \( \Sigma_E \) of integrity constraints, i.e.

\[
E \triangleq (\text{attr}(E), \Sigma_E).
\]

The following types are examples of entity types in Figure 1:

\[
\begin{align*}
\text{Person} & \triangleq (\{\text{Name}, \text{Login}, \text{URL}, \text{Address}, \text{Contact}, \text{DateOfBirth}, \text{PersNo}\}) \\
\text{Course} & \triangleq (\{\text{CourseID}, \text{Title}, \text{URL}\} \cup \{\text{ID}(\{\text{CourseID}\})\} ), \\
\text{Room} & \triangleq (\{\text{Building}, \text{Number}, \text{Capacity}\} \cup \{\text{ID}(\{\text{Building}, \text{Number}\})\} ), \\
\text{Semester} & \triangleq (\{\text{Term}, \text{Date(Starts, Ends)}\} \cup \{\text{ID}(\{\text{Term}\})\} ).
\end{align*}
\]

An ER schema may use the same attribute name with different entity types. For instance, the attribute \( \text{URL} \) in Figure 1 is used for characterising additional information for the type \( \text{Person} \) and the type \( \text{Course} \). If they need to be distinguished, then complex names such as \( \text{CourseURL} \) and \( \text{PersonURL} \) are used.

Objects on type \( E \) are tuples with the components specified by a type. For instance, the object (or entity) \((\text{HRS3}, 408A, 15)\) represents data for the \( \text{Room} \) 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 $(l_i, o_j)$ for objects $o_j$ 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 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 $PlannedCourse$ component 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 $ProposedCourse$ 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.

$$ProposedCourse \doteq \{ \text{Teacher, Course, Proposal : Kind, Request : Room, Semester, Set2 : \{ Program \}}, \Sigma_{ProposedCourse} \}$$

$$PlannedCourse \doteq \{ ProposedCourse, [Reassigned : Kind], [Reassigned : Room], \Sigma_{PlannedCourse} \}$$

$$CourseHeld \doteq \{ PlannedCourse, [Reassigned : Room], \Sigma_{CourseHeld} \}$$

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.

$$ R \doteq (\text{compon}(R), \text{attr}(R), \Sigma_R). $$

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 IsA between the subtype and the supertype are used. For instance, the type Professor in Figure 1 is a subtype of the type Person. An object (or a “relationship”) on the relationship type $R \doteq (R_1, \ldots, R_n, \{B_1, \ldots, B_k\}, \text{id}(R), \Sigma_R)$ is giving at least $3$ courses and at most $7$ courses. External docents may be obliged by other restrictions, e.g., for courses per semester to at least $0$ and at most $3$, i.e. each course is proposed at most three times in the constraint.

The lookup or look-across constraint

$$ \text{look}(R, R') \doteq m.n $$

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 most three 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 $m \leq \left| \{ o \in R^C : \pi_{R'}(o) = o' \} \right| \leq n$ 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 $R'^{mC}$ may potentially ‘see’ an object $o'$ from $R'^C$. It holds in a relationship class $R^C$ if for any object $o'' \in \text{Dom}(R'^m)$ there are at least $m$ and at most $n$ related objects $o'$ with $\pi_{R'}(o) = o'$, i.e. $m \leq \left| \{ o' \in \pi_{R'}(R^C) : o \in R^C \land \pi_{R'}(o) = o' \land \pi_{R''}(o) = o'' \} \right| \leq n$ for any $o'' \in \text{Dom}(R'^m)$. 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 Teacher, Room, Kind, and Program objects there are between 0 and 7 Docent and Semester.
dependency. The types

The first constraint does not restrict the database. The second constraint expresses a key or functional

DD is called schema if the relationship and cluster types use only the types from

A set

The schema is based on a set of base (data) types which are used as value types for attribute types.

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, ..., E_n, C_1, ..., C_l, R_1, ..., 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, ..., E_n, C_1, ..., C_l, R_1, ..., R_m\} as components and cluster and relationship types are properly layered.

An EER schema is defined by the pair \(\mathcal{D} = (\mathcal{S}, \Sigma)\) where \(\mathcal{S}\) is a schema and \(\Sigma\) is a set of constraints. A database \(\mathcal{D}^C\) on \(\mathcal{D}\) consists of classes for each type in \(\mathcal{D}\) 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 \( V = \{ S_1, ..., S_m \} \), an auxiliary schema \( A \) and a (complex) query \( q : D \times A \rightarrow V \) 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 \( card(R, R') = (m, n) \) 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 \( look(R, R') = m..n \) 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/~thalmheim/HERM.htm
http://www.is.informatik.uni-kiel.de/~thalmheim/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/~thalm/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 \( P \) (powerset) for subset, product and nesting are complete for the construction of sets. Subsets 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