Advanced Content Management

The Kiel Approach to Content Management: Content through Media Types, Concepts through Theories, Metadata and Annotations through Topics, Information through Memes
Collection of Recent Papers

Bernhard Thalheim

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

1 Advanced Content Management

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.

1.1 Media Types and Content

The structural part of content 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. The structure dimension of content chunks is based on the theory of media types. Media types combine views and their functions into one type. Media types may be linked to other media types. For instance, we may distinguish input data for the workflow, retrieval data for the workflow, output data of the workflow, display data suites for each stage of the workflows, and escorting data supporting the understanding of each stage of the workflow. Media objects may be structured, semi-structured, or unstructured by the media types. They are data that are
generated from underlying databases, ordered, hierarchically representable, tailorable to various needs and enhanced by functionality for its usage. Since users have very different needs in data depending on their work history, their portfolio, their profile and their environment media types are packed into containers. Containers provide the full functionality necessary for the application and use a special delivery and extraction facility.

1.2 Concepts

The intention dimension of content management is based on concepts. They are the building blocks in human thinking and reasoning, and as such highly flexible. They can be general or specific, concrete or abstract, natural or technological, artistic or scientific. They can apply to things that are real or imaginary. They provide a help for distinguishing between things we observe in the world, or ideas such as truth and falsity, appearance and reality and continuity and discontinuity. Abstract concepts are useful for characterisation of observations, thoughts and expressions. Typical abstract concepts are truth and falsity, sameness and difference, wholes and parts, subjectivity and objectivity, appearance and reality, continuity and discontinuity, sense and reference, meaningful and meaningless and problem and solution.

1.3 Topics Types

The topic part of content 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 provide the general data structure for a user-dependent view on content on the conceptual level.
## Table of contents

1. The Conceptual Framework To User-Oriented Content Management
   published in [Tha07a, Tha08a, FT07]
   related papers: []
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

2. Towards Linguistic Foundations of Content Management
   long version of the paper published in [FT04]
   related papers: [Tha00, ST08d, ST08a]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

3. Facets of Media Types
   published in [ST08c]
   related papers: [Tha00, ST04a, MSTZ04, ST08a]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

4. Engineering Database Component Ware
   published in [Tha07c]
   related papers: [ST06a, BST06, Tha02, Tha04, Tha03]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

5. Semantics in Data and Knowledge Bases
   published in [ST08e]
   related papers: [Tha08b, Tha00, Tha08c, Tha07b, Tha08a]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

6. Towards Semantic Wikis: Modelling Intensions, Topics, and Origin in Content Management Systems
   published in [FT08] and [FT09] (final version)
   related papers: [JTK+08, FFKT06, FCF+06, KBF+05]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

7. Development of Collaboration Frameworks for Distributed Web Information Systems
   published in [ST07a]
   related papers: [Tha07b, Tha08a, ST06a, ST04b, ST05, FRT05]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

8. The Enhanced Entity-Relationship Model
   published in [Tha09]
   related papers: [Tha00, Tha07b, ST08e, ST08a, Tha07d, DMT07, MDT04]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

9. Generalisation and specialisation
   published in [Tha09]
   related papers: [Tha00, BT07]
   see also my website under miscellaneous or Verschiedenes: talks 2007/2008

10. Abstraction
    published in [Tha09]
    related papers: [Tha00]
    see also my website under miscellaneous or Verschiedenes: talks 2007/2008
3 The Kiel Approach to Content and Knowledge Management

Knowledge is nowadays characterised through (1) its content, (2) its concepts, (3) its annotations or topics, and (4) its understanding by the user. Knowledge pieces cannot be considered in an isolated form. For this reason we imagine to use knowledge chunks as a suite of knowledge pieces consisting of content, concepts, topics and information. These dimensions are interdependent from each other. Figure 1 displays the knowledge space.

![Knowledge Space Diagram](image)

**Fig. 1.** The four dimensions of the knowledge space: Data dimension through content, foundation dimension through concepts, language dimension through topics, user dimension through information

**Content and Media Types: The Data Dimension** Content is complex and ready-to-use data. Content is typically provided with functions for its use. Content can be defined on the basis of media types. Content management systems are information systems that support extraction, storage and delivery of complex data.

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., [Sow00] 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, [W3C04b]) while the Web Ontology Language OWL ([W3C04a]) 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 [Dub07]) for editorial content.
Concepts and Theories: The Foundation Dimension  Content may be enhanced by concepts that specify the semantic meaning of content objects. Concepts $C$ are described by the triple 
(meta information, intension specification, extension).

The intension can be specified by providing the logical theory on the basis of a set of formulas of the predicate logics. The extension of $C$ specifies the mappings to the content spaces and is used for associating content with concepts. The concept intension is given by

- by intext of the concept, i.e., a syntactical description of mandatory, normal and optional parameter where ‘optionality’ can be based on different kinds of null or default values and is dependent on time and ‘normal’ parameters are used for properties,
- by context of the application area, of the history of things under consideration,
- by semantics specified through a set of formulas defined over the intext and the context and based on an interpretation theory, and
- by usage and pragmatics that restricts the application and usage of the concept.

Since concepts are rather ‘small’ logical theories we use concept nets for the specification. Concept nets are specified by instantiating a concept definition frame pictured in Figure 2. Concept fields generalize word fields used in [DT02]. Concepts may also be represented by concept diagrams.

![Fig. 2. Concept fields for specification of concepts](image)
Meta-information is used for specification of quality, restrictions of usage, ownership, replacement of concepts, and other associations to concepts.

Concept extension is used for specification of the concept world. Since concept intensions are ‘small’ logical theories we use logical models of concept intensions for concept extensions.

Concepts are the building blocks in human thinking and reasoning, and as such highly flexible. They can be general or specific, concrete or abstract, natural or technological, artistic or scientific. They can apply to things that are real or imaginary. They provide a help for distinguishing between things we observe in the world, or ideas such as truth and falsity, appearance and reality and continuity and discontinuity. Abstract concepts are useful for characterisation of observations, thoughts and expressions. Typical abstract concepts are truth and falsity, sameness and difference, wholes and parts, subjectivity and objectivity, appearance and reality, continuity and discontinuity, sense and reference, meaningful and meaningless and problem and solution. They govern different kinds of human thinking at a fundamental level.

Concepts are vital to the efficient functioning of semantic Wikis. They are organised bundles of stored knowledge which represent an articulation of events, entities, situations, and so on experience. Concepts are necessary for an understanding, for the organisation, for sharing Wikis and for communication. We may assume a simple association between the components of Wikis and concept. The associations may form a complex multi-dimensional network. They may be of specific types such as kind-of, is-part-of, is-used-for and of variable strength. Associations typically correspond to concepts of a more schematic kind than the concepts which they serve to connect.

The classical approach to concepts is based on description of necessary and sufficient criteria for content-concept association. We notice however that most concepts characterising content chunks cannot be captured by means of a set of necessary and sufficient features. Many natural concepts are fuzzy and contextually flexible. Therefore we need to extend the approaches typically assumed for formal semantics to natural semantics. Additionally, the association of content to concepts must not be strict. Some content may be a better example to a concept than other content.

The prototype approach for concept-content association is also limited. Ratings or selections of prototypes are strongly context dependent, e.g., culture dependent and actor dependent. Prototypes are given with certain preference, frequency, sample extraction, learning background, level of verifiability, and under time pressure. The degree of association may vary over time, may be dependent on the concrete usage, and bound by the representation language chosen. Prototype content may also be more or less specific or general for concepts.

Concepts are typically expressed through propositions. The meaning has typically two parts: an element of assertion and something that is asserted. What is asserted is called proposition. The simplest type of proposition consists of an argument and a predicate. Semantical units or propositions are interrelated by entailment. Entailment is different from material implication and relates propositions by forward propagation of truth and backward propagation of falsity. Propositions can be contraries, contradictories, or independent. They may belong to a category or genre of expression, are given in a certain style or manner, are often based on stereotypical norms of expression, de-
pend on ideas and values that are employed to justify, support or guide the expression, reflect aspects of culture or social order, are shaped according to the community that uses them, and are configured by theories or paradigms.

We also may distinguish between the existential approach to meaning based on a correlation of expressions in a language with aspects in the world. The intentional approach associates some kind of representation with concepts as the main constituents of the sense and depends on the cultural context. Whenever content is difficult to interpret then we need to consider concepts, deep structures, unconscious foundations, hidden symbols, annotations or underlying pattern supporting it. If content seems to transparent then we do not need to look for these things. It is often surprising how much background information is necessary for understanding content even such content that appear on the surface to be wholly transparent. There are various connotations and denotations that content may have. We have to consider the arrangements and laws for constructing content phenomena (langue) as well as the various instances that are constructed by constructors and laws (parole). Content can be coded in various ways, e.g. based on different representation such as text or multimedia elements. Content can be differently categorized and organised. We may use conventions that draw on common forms of knowledge. Furthermore, we need to devise different ways for understanding and for association of concepts to content.

To express content syndication information about the concepts behind content has to be stored. The provenance of data was already studied on the instance level ([BKT00,WS97,BDSHW06]) especially for scientific data sets. We can adapt these results for our purposes. We choose a finite set \( C \) from a universe \( U_C \) 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.

We assume \( U_C \) to be a universe of contexts and let \( C \subset U_C \) to be a finite set of contexts. Further, let \( A = \{ A_1, \ldots, A_n \} \) be a set of content chunks. The concept of a knowledge chunk \( C \) is a tuple \((c, A)\) with a context \( c \in C \) where the knowledge chunk \( C \) resides and a set \( A \) of knowledge 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 context dimension also reflects the purpose of the content chunk. Content chunks have a function such as to give an instruction, to control behaviour, to transmit ideas in a memorable form, to enhance social cohesion, or to give users a better understanding. For example, a content chunk that reflect a piece of knowledge may start with a mystery that leads to a conflict situation, may continue with an explanation of the solution or of the discovery as the turning point and may conclude with a res-
olution of the conflict situation. Context [KSTZ03] injection must thus be an integral element for content chunk processing.

Our context model for content chunks extends the usage thesis of [Wit58] that mainly reflect the communication act between a sender and receiver with their intentions, backgrounds, cultures and relationships. Usage context should also consider excluded receivers, value of content chunks to receivers, groups or societies. Content chunks are thus generically [BST06] enhanced by context, refined by intended specifics and instantiated by their specific usage.

Topics and Ontologies: The Language Dimension

Content and concepts may be enhanced by topics that specify the pragmatic understanding of users. 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 [RH05] 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 ([HM03]) 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., \textsc{NEXPTIME} for the restricted OWL-DL dialect, [HPSvH03].) 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 ([HMW05].)

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 ([GG00]). 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. Content chunks are semantically enriched content instances. They are based on 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.

**Information and Memes: The User Dimension** There are several definitions for information.

- The first category of these definitions is based on the mathematical notion of entropy. This notion is independent of the user and thus inappropriate in our project context.
- The second category of information definitions bases information on the data a user has currently in his data space and on the computational and reasoning abilities of the user. Information is any data that cannot be derived by the user. This definition is handy but has a very bad drawback. Reasoning and computation cannot be properly characterised. Therefore, the definition becomes fuzzy.
- The third category is based on the general language understanding of information [SYea03]. Information is either the communication or reception of knowledge or intelligence. Information can also be defined as
  - knowledge obtained from investigation, study, or instruction, or
  - intelligence, news or
facts and data.
Information can also be the act of informing against a person.
Finally information is a formal accusation of a crime made by a prosecuting officer as distinguished from an indictment presented by a grand jury.
All these definitions are too broad.

We are thus interested in a definition that is more appropriate for the internet age.

**Information** as processed by humans,

– is carried by *data*
– that is perceived or noticed, selected and organized by its receiver,
– because of his subjective human interests, originating from his instincts, feelings, experience, intuition, common sense, values, beliefs, personal knowledge, or wisdom,
– simultaneously processed by his cognitive and mental processes, and
– seamlessly integrated in his recallable knowledge.

Therefore, information is directed towards pragmatics, whereas content may be considered to highlight the syntactical dimension. If content is enhanced by concepts and topics, then users are able to capture the meaning and the utilisation of the data they receive. In order to ease perception we use *metaphors*. Metaphors may be separated into those that support perception of information and into those that support usage or functionality.

Users are reflected by actors that are abstractions of groups of users. Pragmatics and syntactics share data and functions. The functionality is provided through functions and their representations. The web utilisation space depends on the technical environment of the user. It is specified through the layout and the playout. Layout places content on the basis of a data representation and in dependence of the technical environment. Playout is based on functionality and function representations, and depends on the technical environment.

![Fig. 3. Dimensions of understanding messages](image)

The *information transfer* from a user $A$ to a user $B$ depends on the users $A$ and $B$, their abilities to send and to receive the data, to observe the data, and to interpret the data. Let us formalise this process. Let $s_X$ denote the function user by a user $X$ for data extraction, transformation, and sending of data. Let $r_X$ denote the corresponding function for data receiveal and transformation, and let $o_X$ denote the filtering or observation
function. The data currently considered by $X$ is denoted by $D_X$. Finally, data filtered or observed must be interpreted by the user $X$ and integrated into the knowledge $K_X$ a user $X$ has. Let us denote by $i_X$ the binary function from data and knowledge to knowledge. By default, we extend the function $i_X$ by the time $t_{i_X}$ of the execution of the function.

Thus, the data transfer and information reception (or briefly information transfer) is formally expressed it by

$$I_B = i_B(o_B(r_B(s_A(D_A))), K_B, t_{i_B}).$$

In addition, time of sending, receiving, observing, and interpreting can be taken into consideration. In this case we extend the above functions by a time argument. The function $s_X$ is executed at moment $t_{s_X}$, $r_X$ at $t_{r_X}$, and $o_X$ at $t_{o_X}$. We assume $t_{s_A} \leq t_{r_B} \leq t_{o_B} \leq t_{i_B}$ for the time of sending data from $A$ to $B$. The time of a computation $f$ or data consideration $D$ is denoted by $t_f$ or $t_D$, respectively. In this extended case the information transfer is formally expressed it by

$$I_B = i_B(o_B(r_B(s_A(D_A), t_{s_A}), t_{r_B}, t_{o_B}), K_B, t_{i_B}).$$

The notion of information extends the dimensions of understanding of message displayed in Figure 3 to a web communication act that considers senders, receivers, their knowledge and experience. Figure 4 displays the multi-layering of communication, the influence of explicit knowledge and experience on the interpretation.

The communication act is specified by

- the communication message with the content or content chunk, the characterisation of the relationship between sender and receiver, the data that are transferred and may lead to information or misinformation, and the presentation,
- the sender, the explicit knowledge the sender may use, and the experience the sender has, and
- the receiver, the explicit knowledge the receiver may use, and the experience the receiver has.

![Communication Act Diagram](image-url)

*Fig. 4. Dimensions of the communication act*
We approach the analysis of WIS usage as the first important part of storyboarding pragmatics. WIS usage analysis consists of three parts:

1. *Life cases* capture observations of user behaviour in reality. They can be used in a pragmatic way to specify the story space. The work on life cases was reported in a previous publication [ST07b].

2. *User models* complement life cases by specifying user and actor profiles, and actor portfolios. The actor portfolios are used to get a better understanding of the tasks associated with the WIS. The work on user models was reported in a previous publication [ST06b].

3. *Contexts* complement life cases and user models by characterising the situation in which a user finds him/herself at a certain time in a particular location. We classify various aspects of contexts related to actors, storyboard, system and time, which make up the context space, then analyse each of these aspects in detail. This is formally supported by lifting relations.
References


The Conceptual Framework To User-Oriented Content Management

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

Abstract

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

1 Web Content Management

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

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

CMS and web CMS specifically support a variety of tasks:

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

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

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

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

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

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

Therefore, we claim that CMS must

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

Content is complex and may become ready-to-use information. Information is related to the users dimension. Information as processed by humans, is data perceived or noticed, selected and organized by its receiver, because of his subjective human interests, originating from his instincts, feelings, experience, intuition, common sense, values, beliefs, personal knowledge, or wisdom simultaneously processed by his cognitive and mental processes, and seamlessly integrated in his recallable knowledge. CMS are information systems that support extraction, storage and delivery of complex information. Thus, we claim that content specification must use specification of structuring, functionality, distribution, and interactivity. The co-design approach [Tha00, Tha03] presented in this paper may be used for specification of content structure and of content workflow.

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

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

2 Separating Content into Media Objects, Concepts, and Topics

The broad variety of definitions of CMS and the disagreement on a common definition requires to briefly introduce our understanding of CMS and content systems. It is based on the requirement that a content management system must be backed by an information system. Content is often considered to be a generalization of knowledge, information, and data. This generalization must capture all aspects of concern. We separate three different aspects of concern for content and CMS: syntactical aspects mainly related to data management, semantical aspects mainly related to the knowledge background, and pragmatical aspects mainly related to the utilization, annotation, and querying of users and user communities. Instead we prefer a separation of aspects of concern:

- **Pragmatics** concentrates on the meaning of terms used by a user.
- **Semantics** expresses the interpretation of terms used by a community of users.
- **Syntax** restricts attention to the language, its construction, and the way of using it through utterances.

This separation is expressed in the semiotic triangle in Figure 1. Media objects [ST04] are associated with concepts that specify the semantical meaning of media object suites and topics that specify the pragmatical understanding of users. Media objects are data that
are generated from underlying databases, ordered, hierarchically representable, tailorable to various needs and enhanced by functionality for its usage. Concepts are small theories representing the meaning of content. Topics include the annotation of content. The underlying theories are either theories based on information systems, or on mathematical logics and on concept theory, or on semiotics and corresponding logical theories.

**A content system** [Tha04b] consists of a content management system and a set of media object suites, of concepts and topics. The CMS uses special subsystems for management of media objects, concepts and topics. The first subsystem is an extended database management system. The concept subsystem has features for export and import of concepts, for recording and archiving concepts, for distributing concepts, for sharing concepts, for quoting and reusing concepts, and for editing fragments of concept suites. Therefore, this subsystem can be understood as a specific knowledge base [Tan03]. The topic subsystem supports functions for merging a topic into a topic map, merge base names, merge a topic with another topic, and merge a topic map with another map.

Media objects may be structured, semi-structured, or unstructured. A *suite* consists of a set of elements, an integration or association schema [Tha04c] and obligations requiring maintenance of the association [Tha00, Tha03]. In the case of a media object suite, we specify media objects based on a type system enabling in describing structuring and functionality of media objects, in describing their associations through relationship types and constraints. The functionality of media objects is specified by a retrieval expression, the maintenance policy and a set of functions supporting the utilization of the media object and the media object suite.

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

- **interpretation** that maps concepts to content suites,
- **foundation** that provides concepts for given content suites,
- **explanation** that maps topics to concepts,
- **presentation** that relates topic suites to content suites,
- **annotation** that represents content suites by topics, and
- **content delivery** that provides content suites for given topic suites.

**Figure 2**: The mappings of the syntax, semantics, and pragmatics dimensions
Media object suites are presented by possible databases, i.e. the data world. The representations may very depending on the model used. Concept worlds are represented by theory worlds. The modeling world depends on the logical theory used for the representation. Topics are used to represent the user world. Topic suites may be represented by ontologies, taxonomies, dictionaries, or glossaries. They are used for communication among users. Therefore, topics are based on a vocabulary a users group has agreed upon. Media object management is based on a database and computation environment. Concept management is based on model theory. Topic management uses a presentation, visualization, and language environment.

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

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

**AI and theorem proving systems** that enable in deriving new pieces of concepts and that support handling of small logical theories, and

**topic or ontology engines** which are based on XML technology, name spaces, linking facilities.

The mappings interpretation, foundation, explanation, presentation, annotation, and (content) delivery can be developed using classical Discrete Mathematics or database theory [Tha00]. The concept-media object query facility in [TV02] shows that delivery can be based on the product of explanation and interpretation. The association between media objects and concepts can be defined through queries added to each concept triple $C$ [FT04]

(meta information, intension specification, extension)

and the **media type schema** defined on a database schema $S$ and query $q$ defining the content depending on a database state.

**Topics** $\mathcal{T}$ are described by the triple

(user community, topic description, topic population)

for a given user community (or cultural context based on a population that serves as typical examples for the given topic. The topic description is given by

(topicRef, subjectIdentity, scope, baseName, association, roles, member, parameters).

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

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

\(^1\)A typical thesaurus is the dictionary developed by wikipedia community groups. The entries in wikipedia are agreed within a certain community but neither validated nor integrated into a common theory.
3 The Referent or User Dimension Extending The Semantic Triangle

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

3.1 The Referent or User Dimension for CMS

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

Humans have their reasoning capabilities, their memory chunks, and their expression capabilities. The memory chunks should be based on the achievement of neural network research. So far, it is assumed that humans based their reasoning and storage on suites of neurons. These suites can be called memes that are specified by

- names or (fuzzy or navigation) identification facilities,
- a number of properties,
- a variety of associations with different co-/adhesions and repulsion to other memes,
- a variety of activation and deactivation facilities,
- and a variety of groupings for different purposes.

In Figure 3 we extend the semantic triangle by the user dimension and relate memes to topics based on user understanding, user enrichment, and user expression capabilities.

Figure 3: Extending the semiotic triangle to a tetrahedron for CMS by the referent or user dimension
This notion generalizes the notion of knowledge objects developed for knowledge maps. Memes are related to their users. We follow the approach of [Cho82] and use a two-step procedure similar to [BST06] for memes evolution.

Memes are extensively discussed and applied in [Bla99, Tan03]. These resources consider memes to be units of cultural evolution and selection. They can be folded and be used for derivations. The main operations on memes are understanding, enrichment, and expression. These three kinds of operations are similar to the main database operations: read, compute, and write. We extend these operations to general transformations: replication operations depending on replication slots, operations for extracting, transforming and loading memes into other memes, and composition operations.

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

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

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

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

The semiotic triangle has been extended to a tetrahedron in Figure 3. We may now view this tetrahedron from the user point in Figure 4 based on tripods. The user bases his understanding on media objects, concepts, and topics on the basis of views, chunks, and utterances, respectively. We additionally consider the data schema and the data necessary for describing media

![Figure 4: The mappings from and to the user dimension](image.png)

Figure 4: The mappings from and to the user dimension

types, concepts, topics, and memes. The left tripod in Figure 4 describes the schemata used for specification of the media types, concept, topic and meme worlds. The right tripod shows the corresponding data suites used for each layer of concern: media types, logical theories, topic landscapes, and user memes.
3.2 Brain-Pattern, Memes, and Information

We use a notion of information that is better fitted to the needs of information systems and of CMS. It bases the existence of information for a user on this user's abilities for perception (1), abilities for selection (2), the interests (3), the knowledge obtained so far (4), and abilities for integration (5). This notion nicely corresponds to different uses of information as noted in [Tan03]: generation, externalization, recording, protection, communication, distribution, sharing, referencing, editing, search, analysis, management, and annihilation. This notion of information does not directly lead to a definition similar to the definition of information based on the entropy information or of information based on the logical and derivational power. It is, however, user-centered.

So far no commonly accepted theory of meme structures of the brain has been proposed. Classically memes have been considered as structures that are encoded within a gene or a suite of genes. We may, however, consider also dynamic memes that allow a change over the lifespan of a human organism. To develop this, we use biochemistry [RPZ06] and introduce the brain-pattern (b-pattern) that consist of a suite of neurons. B-pattern may be stable or instable. Their formation, transformation, and removal requires energy. Therefore, stability and transformation is based on minimal energy consumption. Additionally, b-pattern have their compositionality and replication that is characterized by

- the general ability for composition or replication, e.g., characteristics describing when composition may appear,
- the general properties required for composition or replication, e.g. the free slot property describing whether composition may take place for a part of the b-pattern and the support for agglomeration,
- the topological and geometrical properties including distance and relative location, and
- the ad-/cohesion and repulsion within a suite and to other suites.

Repulsion and cohesion allow to describe the energy that is needed for composition. Replication of b-pattern is based on folding donors. Given two memes $xDy$ and $x'Dy'$ with a donor. Then the memes may be crossed at the donor to $xDy'$ and/or $x'Dy$. Crossing may lead to the death of one of the results if the energy level is too low. A specific kind of crossing is the linked transformation of a meme to a background meme.

Using b-pattern we may now characterize the meme as a suite of b-pattern that is enhanced by activation facilities. These facilities support building, removal, and change of memes. Typical activation facilities are based on stimulants such as positive or negative emotions, good or bad practices (for keeping, refining or checking), and requests for change or replacement (delete, store in the background and link, insert, update). Stimulants are usually increasing or decreasing the energy level. Requests for change are often based on imposing some stress to memes. If the energy level becomes too low for a meme then this meme is lost or forgotten. This process can also be explicitly described by deactivation mechanisms.

Memes can be accessed by pattern matching or by navigation. Pattern matching is based on overlay structures that might be applied. The access to a meme may lead to a change of pattern (control memes), to invocation of replication or composition (collector memes), or to orchestration of a new set of b-pattern (information meme). Navigation is based on a number of different facets [Tha00]. We may now combine composition, replication and query functions to complex functions that represent the users ability for deduction, induction, abduction,
and reasoning such as non-monotonic, approximate, temporal, epistemic, and qualitative reasoning. The limited ability of users to apply formal reasoning, their specific kind of logics, their specific topic landscape can also be represented through suites of memes.

3.3 Utilization of User Profiles For Advanced CMS

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

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

Figure 5: Kiviat graphs representing spectra

User profiles characterize users by user preferences such as

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

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

preferences for dialogues such as dialogue properties, dialogue forms and styles, dialogue structuring, dialogue control, and dialogue support necessary;

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

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

2The profiles may range from pupils or pensioners interested generally in something to well-informed, educated, critical users seeking additional well-specifiable information. The portfolio may range from inhabitant through tourists to business people seeking special information for their current tasks.
knowledge of the user, e.g., application knowledge depending on application type, application domain, application structuring, and application functions;

privacy restrictions users apply to partners, to general public, or to specific content;

task knowledge, especially task expertise and task experience;

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

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

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

3.4 Natural Language Foundation For User Portfolio

A portfolio consists of

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

The task is given by [Pae00]

• a specification of current and target states,
• a characterization of goals of the task,
• a number of operations that might be used to achieve the task,
• a metrics for evaluation of distance from the target state and the progress of completion,
• a characterization of knowledge necessary for completing the task, and
• a set of control frames characteristically used for completion of the task.

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

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

User express their questions, their update requirements, and their input or deletion on the basis of natural language utterances relating memes to topics, their understanding of chunks

\(^3\)Actors are abstractions of groups of users that have the same intention and share goals.
of logical theories and their views on the media objects. The development of user functionality may be based on the narrative expressibility of users. This expressibility is based on natural languages. In Indo-European languages verbs express activities. Activities of users may be characterized by verbs of action [Hau00] such as buy, learn, and inform, ergative verbs such as escape, process verbs such as fall asleep (ingressive verbs) and wither (regressive processes) and verbs describing a state such as sleep or have.

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

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

The first eight groups are relevant for CMS and may be used for functionality development.

The functionality of advanced CMS may be based on discourse types known from conversation theory:

**Actions**: The partner is requested to do something.

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

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

**Orientation**: An orientation for the next actions of the partner is provided.

We can, thus specify a CMS portfolio of the user by and algebraic expression of SiteLang with the basic portfolio elements given by

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

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

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

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

**Execution model** to be applied for the user step.
3.5 Handling The Vast Variety of Usage Invocations

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

*collecting the actual profiles and portfolio* of current users depending on the actual usage,

*integrating the actual usage into the usage star* that consists of a combined profile and a suite of interrelated portfolios, and

*assembling the topic landscape based on usage stars* by associating the user memes to those topics that correspond to user communities which work on tasks that are related to tasks within the usage star and that are supporting users of the profile that is valid for the current user and collected within the usage star.

Given a user $u$. The user $u$ has a profile or a number of profiles which can be combined through nesting into $\text{CurrProfile}(u)$. Furthermore, given a set $\text{CurrPortfolio}(u)$ of current portfolio of this user. We may now use the star type $\text{UsageStar}(u)$ [Tha04a] that combine the common properties and tasks of the user $u$ and the portfolios.

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

3.6 Approaches for Coping With User Understanding and Abilities

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

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

- Users use denotations for representing their observations and belief on the reality. These denotations can be mapped to variables. The signification (intension and comprehension)

---

4Depending on the profile specification we may assume that the current profiles of a user are given by a set of ER objects that may be combined into a complex nested object. The operations developed for advanced ER models (e.g., join, product, unnest, nest, rename, difference, set operations [Tha00]) may be used to define the profile combination operations.
and the meaning (reference, étendue) of these variables may vary depending on the user world and the user memes we are considering.

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

- Identity, existence and identification vary in users world.

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

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

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

- User may use qualitative reasoning instead of logical reasoning.

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

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

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

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

- Usage of memes may depend on the context, on the auditory, on the purpose and other environmental parameters.

This variety may be considered to be the playground of logicians.

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

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

$$\forall x (P(x) \implies Q(x)), P(a) \implies Q(a)$$

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

**Reasoning based on induction** uses a background theory $\mathcal{B}$ and observational data $\mathcal{D}$ with the limitation $\mathcal{B} \not\models \mathcal{D}$. It is based on such for a formula $\alpha$ that is consistent with the data ($\mathcal{B} \cup \mathcal{D} \not\models \neg \alpha$) and explains the data ($\mathcal{B} \cup \{\alpha\} \models \mathcal{D}$).
Abductive reasoning allow to derive explanations $E$ within a set of hypotheses $H$ ($E \subseteq H$) for observations $O$ on the basis of a logical theory $\Sigma$, i.e. we seek for a set $E$ through which a user may explain the observations $\Sigma \cup E \models O$. We may require that the set $\Sigma \cup E$ is consistent. Rules such as the pseudo modus ponens $\forall x (P(x) \Rightarrow Q(x)), Q(a) \Rightarrow P(a)$ or the modus tollens $\alpha = \beta, \beta = 0 \Rightarrow \alpha = 0$ may be used.

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

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

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

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

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

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

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

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

At the same time, we characterize the languages we use for representation of memes and for reasoning based on memes by different layers of adequacy:

- **Epistemic adequacy** characterizes the expressive strength of the language used.

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

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

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

Users reasoning abilities are characterized by their
logical language that is used for representation of memes, for associating memes by connectives and quantifiers, and for constructing formulas on memes,

reasoning procedures such as combined inductive reasoning and qualitative reasoning, and

their abilities to cope with inconsistencies on the basis of knowledge islands.

4 Development of Systems Supporting User-Oriented Content Management

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

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

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

Well-foundedness: A CMS is well-founded if the two subset properties
interpretation(explanation(t)) ⊆ delivery(t)
and
presentation(foundation(cs)) ⊆ annotation(cs)
are valid for any topic t and any media object suite cs.

Faithfulness: A user-oriented CMS is faithful if
interpretation(explanation(associate(m))) ⊆ delivery(associate(m))
for any meme m.

Saturatedness: A CMS is saturated if
interpretation(explanation(t)) ⊇ delivery(t)
and
presentation(foundation(cs)) ⊇ annotation(cs)
are valid for any topic t and any media object suite cs.

Consistency: A CMS is consistent if
interpretation(explanation(associate(m))) ⊈ delivery(associate(m))
for any meme m.

Based on these properties we need to solve the following problems:

Foundation problem: A CMS is well-founded if no topic exists that may be associated with a concept or a concept set which are associated to media objects which are not annotated by the given topic. So, the foundation problem consists in association of all topics which are not well-founded.

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

Faithfulness problem: The system becomes faithful if all memes of users are represented by faithful topics. So, the problem consists in finding those memes which do not have an association to founded topics.
Consistency problem: We need to detect those memes that are not associated to saturated topics and then to repair this inconsistency.

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

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

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

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

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

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

This set of problems may be considered as open problems. Similar to [Tha04a] we may however base profiles and portfolio on multidimensional characterizations that can easily be combined.

4.2 Functions Mapping Between Memes And Concepts, Media Objects, and Topics

The three additional structures beside infons, assets and units are chunks associating concepts with memes,

utterances associating topics with memes, and

views associating media objects with memes.

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

We must now consider a number of different views:

- The user understands chunks of concepts.
• The user **expresses** data needs through **utterances** based on association to topics.

• The user **queries** for media objects or data through **views**.

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

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

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

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

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

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

The **portfolio manager** allows to derive, to manage, to change, to retrieve and to associate portfolio of the user. The portfolio manager may use a specific task glossary that supports analysis of the meaning of utterances.

The **profile manager** supports storage, retrieval, change, and introduction of user profiles. User profiles may include specific slang-like vocabularies.

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

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

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

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

![Figure 7: Proposal for an architecture of user-oriented content management systems](image)

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

The architecture neatly integrates the conceptions for content and the user worlds. Information is then representable either by the pair (meme, concepts) or by the pair (meme, topics landscape) or by the pair (meme, media objects) or finally by the pair (meme, content).
5 Conclusion

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

separating concerns in dimensions for data, logical foundations and representational (topic) worlds,

handling each of the dimensions separately by providing sophisticated functionality for the dimension,

adding the user worlds through explicit representation of their understandings, and

mapping facilities between the syntax, semantics, pragmatics and referent dimension.

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

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

Acknowledgement. The author is thankful to N. Lukashev (Lomonosov University Moscov) for discussions and the explanation of (bio)chemical foundations behind [RPZ06].

References


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

Gunar Fiedler and Bernhard Thalheim

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

Abstract

Content and content management have become buzzwords. The notions are neither well-defined nor used in a standard way. Content objects are complex structured. Different users may require different content object sets. Content object sets may vary depending on the actual task portfolio, depending on the context of the user, and on the technical environment. Therefore, content management must combine generation, extraction and storage of complex object, must support complex workflows and must be adaptable to the actual use and users environment and requirements. Content may be considered under three different points of view: data computed from an information system, general concepts that are illustrated or described by the content, and, finally, a large variety of user interpretation. Since all three points of views should not be mixed with each other we propose to separate them and treat content management from the point of view of syntax, from the point of view of semantics and from the point of view of user worlds.

1 Content and Content Management

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

A wide variety of systems claim to be a content management system (CMS), e.g. CacheWare, ConnectSite.com ASP, ContentPlanner, Coremedia, Corevue, Documentum, DynaBase, E-Grail web management platform Ektelon, eKeeper, ECOMS, Eprise, Gauss, Imparto Web Marketing Suite, Intervwowren, IntraNet Solutions, iDB Browsinform, iMakeNews.com, Midgard, NCompass, OnDisplay, SiteC, SiteDriver, SiteGeneral, SiteManager, SiteMerger, SiteStation, Stage2Live,

* Long version of: Towards linguistic foundations of content management. Proc. NLDB’2004, LNCS 3136, 348-353
Vignette, Website ASP, etc. There are surveys [Jou03] that keep lists of content management systems. Roughly we may classify content management systems into website content management systems, enterprise content management systems, advanced document management systems, and extranet content management systems. This large variety of systems has a number of properties in common: generation, delivery and storage of complex structured objects; rights management; service management in distributed environment; customer management; update and quality management; context dependent delivery depending on the user, the HCI, and the actual systems situation.

The content of a CMS is a most value asset. Content must be updated frequently to keep user coming back and to succeed in their tasks. Thus, a content management system supports production of content while automating some of the frequent operational tasks.

CMS and web CMS specifically support a variety of tasks:

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

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

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

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

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

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

CMS is not an out-of-box solution. CMS are complex information systems that integrate a number of facets:

Content is structured in a large variety depending on different users, environments and a tasks.

Complex workflows and tasks should be supported.

Distributed services are integrated.

Adaptation and delivery is supported.

Therefore, we claim that CMS must

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

2 Semiotic Separation of Information into Content,
   Concepts and Topics

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

**Pragmatics** concentrates on the meaning of utterances.

**Semantics** expresses the interpretation of utterances used in a language.

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

This separation is expressed in the semiotic triangle in Figure 1.

We may distinguish information and content. Information as processed by
humans, is data perceived or noticed, selected and organized by its receiver,
because of his subjective human interests, originating from his instincts, feel-
ings, experience, intuition, common sense, values, beliefs, personal knowledge,
or wisdom simultaneously processed by his cognitive and mental processes, and
seamlessly integrated in his recallable knowledge. Information is content com-
bined with the interpretation and the understanding of its users. We can now
use this separation of aspects of concern to develop a semiotic understanding of
concepts and information. This separation is displayed in Figure 1. We use the

Fig. 1. Linguistic Separation of Concern Applied to Content Theory

word “content” in a restricted form and mean elements or subsets of business
data. The word “concept” is used for theoretically based modeling worlds. It
is thus used within the semantic or logical theory that forms the basement for
concept worlds. Content and concepts may be denoted in a large variety by dif-
ferent users. Each user has his/her own terminology. We use the word “topic”
for denotation of concepts or content. Therefore we distinguish between three different “worlds” of users:

Considering the **content world** or *representation world* we concentrate our attention to the data and their representation.

Considering the **concept world** we concentrate our investigation to the logical foundations of the content world and of the topic world.

Considering the **topic world** we are interested in the terminology or in the ontology of users, their common understanding.

There are tight relationships between these worlds. Users have their own understanding of their topics and may associate different content with the same topic.

Since we are targeting in a general specification framework for content systems we will not deepen the association to concepts and topics. Defining this theory of CMS we need to solve a number of additional problems:

**Definition of content, concepts and topics** in a well integrated form.

**Development of mappings** between topics, concepts, and content.

**Development of a supporting infrastructure** providing data for content, association of content and concepts, and delivery of content for topics users are interested in.

**Development of integrated functionality** for content management, concept management, and topic management that can be mapped between layers discussed in the next section.

**Development of a modeling language** supporting consistent treatment of topics, concepts and content.

In the next sections we develop a program for the solution of these problems.

### 3 Definition of Concepts

**Concepts** $\mathcal{C}$ are described by the triple

$(\text{meta information}, \text{intension specification}, \text{extension})$.

The intension can be specified by providing the logical theory on the basis of a set of formulas of the predicate logics. The extension of $\mathcal{C}$ specifies the mappings to the content spaces and is used for associating content with concepts.

The **concept intension** is given by

- **intext** of the concept, i.e., a syntactical description of mandatory, normal and optional parameter where ‘optionality’ can be based on different kinds of null or default values and is dependent on time and ‘normal’ parameters are used for properties,

- **context** of the application area, of the history of things under consideration,

- **semantics** specified through a set of formulas defined over the intext and the context and based on an interpretation theory, and

- **usage and pragmatics** that restricts the application and usage of the concept.
Since concepts are rather ‘small’ logical theories we use concept nets for the specification. Concept nets are specified by instantiating a concept definition frame pictured in Figure 2. Concept fields generalize word fields used in [DT02].

Meta-information is used for specification of quality, restrictions of usage, ownership, replacement of concepts, and other associations to concepts.

Concept extension is used for specification of the concept world. Since concept intensions are ‘small’ logical theories we use logical models of concept intensions for concept extensions.

For instance, the concept of an address supporting in enterprise information systems some tasks of human resource management consists of

\[\text{Address}((\text{geographAddr}, \text{contactAddr}, \text{history}), \left(\sum_{\text{PL} \times 1}^{\text{Address}}, M_{\text{Address}}, \Sigma_{\text{Quality}}\right), \left(\text{enterprise}_{\text{IS}}, \text{tasks}_{\text{of}_{\text{HRM}}}\right))\]

where \(\Sigma_{\text{Address}}\) specifies the small logical theory of addresses.

Concepts may also be represented by concept diagrams. A typical example representing the concept of Person is displayed in Figure 3.

Optional components of concepts are either displayed through an explicit presentation with the \([\,)\] constructor or relationship types. Kernel concepts are underlined. Normal concepts are displayed without any adornments. Parameters of concepts are abstract. They are displayed with an underlined prefix. They are instantiated to concrete types of content. Normal concepts or mandatory
concepts can also be represented by relationship types with assigned cardinality constraints.

The concept of Person is based on formulas $\Sigma^\text{DeontTempPL}/1$ from temporal deontic predicate logic. An example of such formula is the formula $F(\text{update(Person._Birth.data)})$ stating that it is forbidden ($F$) to change birth data of a person. This formula illustrates the difference between data and concepts. The birth date in a database may be changed under certain conditions. The constraint

$$\alpha \text{“divorced”(person)} \rightarrow \exists_{\text{past}} y$$

$$(\text{Association(is.Partner.y,of.Partner.person,since,until, Kind.descr})$$

$$\wedge \text{descr = “married”} \wedge \text{until < today} \wedge y \neq \text{person}$$

states that for people which are divorced we need information on previous partners. Variables are depicted by small letters.

We can generalize the theory of word fields to a theory of concept fields. The mind map of concepts in Figure 2 is based on the theory of word fields [Kun92,SSS90]. A word field is a linguistic system in which similar words

– that describe the “same” basic semen
– that are used in the same context

are combined to a common structure and data set. In contrast to common synonym dictionary, word fields define the possible/necessary actors, the actions and the context. Word fields can be used for verbs, nouns and adjectives. We focus on verb fields and extend the implementations of the WordNet dictionary for verbs. For each verb field we can at least define the following abstract information:
– basic semen,
– context of usage,
– possible/necessary actors, actions and context, and
– the star or snowflake specification:
  logical structure specifying possible arguments,
  semantic description of relevant and irrelevant valences, association to semantic type (color, space, time etc.),
  kernel semantics describing the word field with semen, and
  example sentences providing additional information usable for help desk systems.

Word fields are well-specified for a number of verbs.
In our generalization, concept fields are different from word forms that carry the grammatical variants. A word is an abstract concept which is concretely manifested solely in the associated word forms [Hau00]. Word fields combine aspects defined in phonology (sounds), morphology (word form), syntax (sentence), semantics (meaning) and pragmatics (use).

4 Definition of Topics

Our topic notion generalizes and formalizes the notion of topics [Pe01] commonly used for topic maps and implemented in [Ont]. Topic maps are based on conceptual structures [Sow00]. Our notion integrates these proposals with the Pawlak information model [Paw73] and concept lattices [GW98].

Topics \( \mathcal{T} \) are described by the triple

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

User characterization may be based on the specification of users knowledge or beliefs, on the personality profile, and on the task portfolio. Topic description \( \mathcal{T}_D \) can be given by the specification of

(topicRef, subjectIdentity, scope, baseName, association, roles, member, parameters).

Topic population \( \mathcal{T}_P \) is specified by the specification frame

(instanceOf, resourceRef, resourceData, variant, occurrence).

We typically require that topic description and topic population are coherent. The coherency notion is based on the information model of Pawlak (see [Tha00]). We are given a universe of discourse \( \mathcal{K} \) of the user \( K_B = (C, U, val) \) with things \( C \) used for the topic population \( \mathcal{T}_D \), properties from \( U \) used for associating things with the topic description \( \mathcal{T}_D \), and the function \( val : C \times U \rightarrow \{0, 1\} \) that associates properties with things.

Topics \( \mathcal{T} \) are coherent if topic descriptions are tightly associated with their populations, i.e., the equalities

\[
\begin{align*}
\mathcal{T}_D &= \{ A \in U \mid \forall o \in \mathcal{T}_P : val(o, A) = 1 \} \\
\mathcal{T}_P &= \{ o \in C \mid \forall A \in \mathcal{T}_D : val(o, A) = 1 \}
\end{align*}
\]

are valid for each pair \( (\mathcal{T}_P, \mathcal{T}_D) \in \mathcal{P}(C) \times \mathcal{P}(U) \) of sets \( \mathcal{T}_P \) of things denoted by a topic \( t = (B, \mathcal{T}_D, \mathcal{T}_P) \in \mathcal{T} \) and the properties \( \mathcal{T}_D \) taken under consideration.
The topics of Address specify addresses by means of geographical address or contact addresses. Typical specific addresses are living address, mailing address, delivery address, and secondary living address. The address topic does neither cover diplomatic addresses nor memory locations in a computer. This topic is associated with the address concept defined in the previous section. The topics of Person is exclusively used for living human beings.

5 Definition of Content

A content system consists of a content management system and a set of content object suites. Content objects may be structured, semi-structured, or unstructured. A suite consists of a set of elements, an integration or association schema and obligations requiring maintenance of the association. In the case of a content suite, we specify content objects based on a type system enabling in describing structuring and functionality of content object, in describing their associations through relationship types and constraints. The functionality of content object is specified by a retrieval expression, the maintenance policy and a set of functions supporting the utilization of the content object and the content object suite.

Our notion extends modern approaches to content suites [SS03] and combines them with the theory of media objects [FKST00]. Classically, (simple) views are defined as singleton types which data is collected from the database by some query.

```
create view name (projection variables)
select projection expression
from Database sub-schema
where selection condition
group by expression for grouping
having selection among groups
order by order within the view
```

Simple examples of a view suites are already discussed in [Tha00] where view suites are ER schemata. The integration is given by the schema. Obligations are based on the master-slave paradigm, i.e., the state of the view suite classes is changed whenever an appropriate part of the database is changed.

The relational view specification frame is generalized to the frame:

```
generate MAPPING : VARS → OUTPUT STRUCTURE
from DATABASE TYPES
where SELECTION CONDITION
represent using GENERAL PRESENTATION STYLE
& ABSTRACTION (Granularity, measure, precision)
& ORDERS WITHIN THE PRESENTATION
& Hierarchical representations
& Points of view
& Separation
browsing definition CONDITION
```
& Navigation
& Search functions
& Export functions
& Input functions
& Session functions
& Marking functions

A content schema $\mathcal{D}$ on a database schema $\mathcal{S}$ consists of a view schema $\mathcal{D}_V$, a defining query $q_\mathcal{D}$, which transforms databases over $\mathcal{S}$ into databases over $\mathcal{D}_V$, and a set of functions defined on the algebra $\mathfrak{A}(\mathcal{D})$ on the content schema.

The defining query may be expressed in any suitable query language, e.g. query algebra, logic or an SQL-variant. For our purposes, however, this is yet not sufficient. One key concept that is missing in the views is the one of link. Therefore, we must allow some kind of “objectification” of values in the query language. A set $\{v_1, \ldots, v_m\}$ of values is transformed into a set $\{(u_1, v_1), \ldots, (u_m, v_m)\}$ of pairs with new created URLs $u_i$ of type URL – more precisely, we first get only surrogates for such URLs. In the same way we may objectify lists, i.e. transform a list $[v_1, \ldots, v_m]$ of values into a list $[(u_1, v_1), \ldots, (u_m, v_m)]$ of pairs. We shall talk of query languages with create-facility.

As a second extension we may want to provide also auxiliary content [FST98]. A extended content type has a name $\mathcal{D}_X$ and consists of a content schema $\mathcal{D}$, a defining query $q_{\mathcal{D}_X}$ with create-facility that defines a view, and a binding between $\mathcal{D}$ and $q_{\mathcal{D}_X}$.

Auxiliary content types are not yet sufficient for information service modelling. The representation of content types and auxiliary content types may depend on the user profile, on the user task portfolio, on units of measures, on representation types, on the order of representation, and finally on the container for delivery [ST01]. Therefore, we use wrapped content types $\mathcal{D}_W$ for representation of content types that are adaptable to users and to the environment.

6 Layering of Content, Concepts and Topics

Figure 4 shows the layering of topic maps, concept worlds, content suites or content bases and databases. At the pragmatics layer people are interested in their topics or topic maps. These maps are associated to theories from the semantics layer. Theories are used to express the understanding through topics. Theories are interpretations of content, i.e., business data. Business data consist of content. Content may be obtained through querying from data and may be conceptualized to concepts.

We observe, furthermore, that different quality criteria may be applied at different layers. For instance, at the pragmatics layer users are interested in utility and useability. Other properties are functionality and understandability.

At the semantic layer, users must be supported by facilities to integrate their knowledge and beliefs with the content they receive. Therefore, consistency of content received with the belief and knowledge, interpretability of content, belief
integrateability for the information received, and communicability or transferability of the content received become an issue of system development. Content must be deliverable at the time requested, in the quality requested, in the format acceptable for the user, displayable at the devices and the environment in use, and depends on the users portfolio and profile. Therefore, ubiquity, scalability, robustness, and adaptability must be supported by the content delivery system. At the database layer, quality requirements such as performance, concurrency, recoverability, security, and portability are used for system evaluation. This list

![Diagram](image)

**Fig. 4.** The Separation of Data, Content, Concepts, and Topics

of quality requirements layers the issues that must be solved for general content management systems.

The layering structure follows classical abstraction concepts. We separate between mini data (such data that are stored in a database), condensed or aggregated macro data (data in a form relevant to a user) called content, the intention of the data called concepts, and the meaning of data expressed through denotations. The separation in Figure 4 follows this layering approach. Denotations
or concepts may be described through words taken from the users ontology or topic map. We prefer the specification through topics.

Layering supports a general understanding of information where we distinguish three abstractions:

- Aspects of information instances are expressed through content types $D$, extended content types $D^E$, wrapped content types $D^W$, topics $T$, and concepts $C$.

- The schemata used for specification of instances are the content schema $S(D_1, ..., D_n)$, the extended content schema $S(D^E_1, ..., D^E_n)$, and the wrapped content schema $S(D^W_1, ..., D^W_n)$, the topic schema $S(T_1, ..., T_m)$, and concept theory $S(C_1, ..., C_k)$.

Similar to database schemata we combine a set $\{T\}$ or sequence of types into a schema $S$ by extension through operations defined on the types and through semantics defining the integrity constraints on top of the schema.

- The instance sets for the different aspects are the content base $S^C(D_1, ..., D_n)$, the extended content base $S^C(D^E_1, ..., D^E_n)$, and the wrapped content base $S^C(D^W_1, ..., D^W_n)$, the topic map $S^T(T_1, ..., T_m)$, and finally the concept world $S^C(C_1, ..., C_k)$.

We use the $R$ adornment for “classes” of instances defined over the corresponding schema or type $R$.

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

A CMS supports the management of content suites. Based on data warehouse architectures, a content management system consists of three sub-systems:

- **Content object extraction system** for extracting, purging and integrating content into the CMS,
- **Content object storage and retrieval system** supporting efficient storage and retrieval of content within the CMS, and
- **Content object delivery system** supporting variable playout of content to users depending on their environment, their needs and their context [KSTZ04].

Content, concepts and topics must be tightly associated with each other. If an algebra for construction, retrieval and manipulation is defined for content suite then such algebra is sought for concept suite as well as for topic suites.

## 7 Computational Support for Content Management

The broad variety of definitions of CMS and the disagreement on a common definition requires to briefly introduce our understanding of content management systems and content systems. It is based on the requirement that a content management system must be backed by an information system. A content system consists of a content management system and a set of content object suites.
It is based on a database management system with the structuring and functionality. The playout is based on the story space specification [TD01], the actor specification [TD01], and the integration of context [KSTZ04].

The general architecture of a content system is shown in Figure 5. It generalizes an architecture to information system that has successfully been applied in more than 30 projects resulting in huge or very large information-intensive websites and in more than 100 projects aiming in building large information systems.

![Fig. 5. Components of Content Management Systems](image)

A content management system, thus, consists of an information system extended by facilities for management of content suites, for user adaptation, for user management, and for expressing the utilization and interactivity. Interactivity specification is based on the notion of story spaces and group of users (called actors). It supports context awareness. The specification of content systems is based on structuring and functionality of data types, on structuring and functionality of content suites, and of the story space. The instantiation of the content system consists of databases, content object suites, scenarios and users. The latter depend on the utilization context.
8 Conclusion

This paper proposes a general theory of content management systems in the broader sense. Content management can be separated into content management based on advanced database systems, concept management, and topic management. We have shown in this paper how content management systems in the broader sense can be built. This approach has already got a partial “proof of concept” in some of our projects. The next step of our research is the development of a general theory of operations and integrity constraints for content management.

References


[ST01] K.-D. Schewe and B. Thalheim. Modeling interaction and media objects. In M. Bouzeghoub, Z. Kedad, and E. Mélais, editors, NLDB. Natural Language

Facets of Media Types

Klaus-Dieter Schewe¹, Bernhard Thalheim²

¹ Massey University, Information Science Research Centre
Private Bag 11 222, Palmerston North, New Zealand
² Christian Albrechts University Kiel, Department of Computer Science
Olshausenstr. 40, D-24098 Kiel, Germany
kdschewe@acm.org thalheim@is.informatik.uni-kiel.de

Abstract. The concept of media type is central to the codesign approach to Web Information Systems (WISs). By means of views on some database schema it permits a separation of global content from local content that is to support particular scenes of a WIS. By means of various extensions such as cohesion, hierarchies, style options and associated operations it enables interface abstraction and adaptivity to users, channels and end-devices. It can further be used for modelling collaboration, session support and contextual information, and by means of an associated dynamic logic provides the basis for formal reasoning about a WIS design. In this paper we give a brief survey of the potential of media types as a very powerful abstraction mechanism for WISs.

1 Introduction

A Web Information System (WIS) is a data-intensive system that is accessible via the world-wide web (WWW). As such the design of such systems has attracted a lot of attention by researchers in the area of conceptual modelling, and various modelling languages and design frameworks have been developed so far [2–4, 6, 7, 15, 16], all having their individual merits but none giving a complete answer, how all the challenges in WIS modelling (see [13]) are addressed.

In this paper we give a brief survey of the potential of the concept of media type, the central notion in the codesign approach to WISs [15]. The naming of the concept relates to the common understanding of “media” as means for mass communication suggesting that the WWW has become such a means. “Types” then refer to the classification and formal abstraction from content and functionality of such communication means.

As such a media type is first an interface abstraction related to and extending the notion of dialogue type introduced in [14]. The rough idea is that the global data content of a WIS is captured by means of an instance of a conceptual database schema, while content needed at the interface is modelled by means of views. As emphasized in [13] this includes the modelling of the navigation structure, which in various other approaches is treated as a separate add-on feature. Furthermore, same as with dialogue types operations expressing the
functionality of the WIS, are coupled with the views. In this way, media types capture form-based interfaces [5, 9] in an abstract way.

In doing so, media types provide a perfect mechanism to refine the scenes and actions of the storyboard that result from analysing and modelling the usage of the WIS [15], thus linking the conceptual model to the identified systems requirements. By means of associated style options [11] they also provide the mechanism for system implementation including a structural part (XML clusters), a behavioural part (scripts) and a layout part associated with page grids.

However, media types are more than just interface abstractions. By means of hierarchical versions they subsume OLAP-like features allowing users to switch between representations at various levels of granularity. By means of cohesion they allow an automatic split of information adhering to the restrictions imposed by channels, end-devices and user preferences. In this sense media types enable derivable adaptivity. Furthermore, as outlined in [12] consistency and personalisation with respect to content and functionality can be achieved by logical reasoning about media types.

Most WISs are centred around the actions of individual users, whereas the need may arise to support collaboration. This can also be captured by media types using associated exchange frames [1] that support cooperation, communication and coordination within user groups. This is related to the fact that media objects, i.e. instances of media types, can be used to capture information about sessions, which are linked to complex scenes in the storyboard.

Finally, media types permit dealing with the often mentioned phenomenon of “loss in cyber space”, which actually is a loss of navigation context. Adapting an idea from Contextual Information Systems [18] we obtain query macros that allow a user to traverse back a path in the storyboard and to explore alternative access paths, respectively [8].

2 Interface Abstraction

In the following let $S$ denote some conceptual database schema. The introduction of media types in [15] was based on the higher-order Entity-Relationship model (HERM) [17], but we already emphasised that the choice of data model is of minor importance. A model (such as HERM) with richer structuring mechanisms will simplify the definition of views, while a poorer model – even if it has the same expressive power – will require more sophistication in the queries that define views.

Furthermore we assume a type system comprising base types and various type constructors, e.g.

$$t = b \mid I \mid (a_1 : t_1, \ldots, a_n : t_n) \mid \{ t \} \mid \{ t \} \mid (a_1 : t_1) \oplus \cdots \oplus (a_n : t_n)$$

would define a type system with not further specified base types $b$, a trivial type $I$, and constructors for records, finite sets, lists and multisets, and disjoint unions. Usually, we assume that among the base types there is a type $URI$ representing abstract surrogates for URIs.
The semantics of such types is primarily defined by their domains, i.e. with each type \( t \) we associate a set of values \( \text{dom}(t) \). For a base type the domain is assumed to be countably infinite. For the trivial type the domain is a singleton set \( \{1\} \). For constructed types we use the usual recursive definitions. Furthermore, we associate basic (fixed) operations with the type system, e.g. projection on record types, union for sets, etc., but we dispense with listing them here.

The type system forms the basis for the definition of content type expressions. We simply extend the type system permitting \( r : M \) to appear in lieu of a base type, with \( r \) and \( M \) being names for a reference and a media type, respectively, both taken from some unspecified countable pool of names. Replacing \( r : M \) by URI results in a proper type, called content representation type.

The core of a media type \( M \) is defined by an interaction type, which comprises a content type expression \( ct(M) \) (with corresponding content representation type \( t_M \), a query \( q_M \) defined on \( S \) with output type \( (id : \text{URL}, value : t_M) \), and a finite set \( Op_M \) of operations – more formal details can be found in [15]. An interaction schema \( I \) is a finite set of interaction types that is closed under references, i.e. whenever \( r : M \) appears in a content type expression \( ct(M') \) with \( M' \in I \), then \( M \in I \) must also hold.

The semantics of interaction types and schemata is defined by means of the defining queries. Take an instance \( db \) of the database schema \( S \) and execute the queries \( q_M \) for all \( M \in I \), which will result in sets \( db(M) \) with values that are actually pairs \((u, v)\) with a URL \( u \) and a value \( v \) of type \( t_M \). We request that whenever a URL \( u' \) appear in such a value corresponding to the reference \( r : M \) in the content type expression \( ct(M) \), then \( db(M') \) must contain a value \((u', v')\). Furthermore, URLs must be globally unique. In this way we obtain an abstraction from the structure at the interface. The value \((u, v) \in db(M)\) represents the content \( v \) at the abstract URL \( u \) including references to other units of content, i.e. including the navigation structure. In its simplest form this is abstraction from web pages, but as we shall see later it can be more than that. As emphasised in [15] and discussed with respect to different query languages, the abstract definition of interaction types requires the use of query languages that are able to create abstract identifiers, i.e. the URLs, and in many cases will need a fixed-point construction.

Operations associated with an interaction type permit updating the underlying database and to open and close interaction objects. For this, we require a signature describing input- and output-parameters as well as a selection type, and an operation body that can be defined by means of usual programming language constructs. The selection type specifies, which part of the content value must be selected as a precondition for the operation being executed. In [9] we illustrated that this form of interaction types, which is more or less the same as the dialogue types in [14] captures form processing in a natural way and on a reasonably high level of abstraction, but it is not limited to this.

For this, consider storyboarding, the precursor of conceptual modelling in the codesign approach to WISs. A storyboard consists of three parts: (1) the story space describing scenes, the transition between scenes, actions associated with
these transitions, and the plot, i.e. the detailed action scheme, (2) the actors
describing roles, right and obligations associated with roles, user profiles, and
preferences associated with these profiles, and (3) the tasks describing meaning-
ful chunks of WIS usage, the actors involved, their goals, the scenes associated
with tasks, and the actions required for task execution. Scenes are hierarchically
organised in the sense that each non-elementary scene gives rise to a scenario,
which is composed in the same way as the whole story space, in particular giving
rise to sub-scenes.

The basic interface abstraction mechanism described above associates a me-
dia type with an elementary scene leading to page abstraction as outlined above.
However, there is no formal reason not to associate media types also with non-
elementary scenes, thus leading to interface abstraction on a higher-level. We
will discuss the benefits of this high-level interface abstraction in the following
sections.

Before doing this, let us first take a look at some other facet of media types.
Media types are extended interaction types, and one of the extensions refers to
layout and playout style options. For the layout the primary question is how the
abstract content of a media object, i.e. a value of type \( t_M \), is to be represented.
Exploiting that types give rise to a lattice, this first leads to a decomposition by
means of a Sperner set of types \( t_1^M, \ldots, t_k^M \) with \( \bigcup_{i=1}^{k} t_i^M = t_M \). Using canonical
projections from \( \text{dom}(t_M) \) to each \( \text{dom}(t_i^M) \) we obtain values \( v_1, \ldots, v_k \) that
jointly represent the content of some WIS unit. For each of the types \( t_i^M \) a style
option specifies how records, sets, lists and multisets are to be represented using
maybe tables, itemized lists, forms or other layout elements, which colours are
to be used, which font families and sizes are to be used, which adornments such
as background images should be added, etc. In the same way the playout of
operations is handled, i.e. how are operation names to appear, how is input from
the user collected using forms or dialogue boxes, how is correctness of data entry
ensured, etc.

With respect to the actual system implementation, the content representation
type of a media type gives rise to a cluster of XML templates, while the defining
query has to be mapped to a query resulting in a cluster of XML documents.
The style options then give rise to translators from XML to HTML, while the
operations give rise to scripts implemented in the preferred scripting language.
In this way, most of the actual implementation of the WIS results from page
generation with the actual content defined by the current database instance.

Furthermore, media types permit the easy use of the container metaphor. By
using this metaphor the content representation type \( t_M \) is only considered as the
type describing the content at the server-side, while the demand at the client
side may be expressed by a different type \( t \). In this case, the meet \( t_M \cap t \) defines
the part of the content that is to be shipped in a “container” to this client. Any
missing part has to be obtained from other media types. Style options on the
client side then define how the content of various containers is to be assembled.
3 Adaptivity

Two other extensions of interaction types (turning them into media types) address the granularity of the represented data, and the adaptivity to users, channels and end-devices. The former one adopts ideas from on-line analytical processing (OLAP), while the latter one addresses splitting and condensation of information. Both extensions exploit the type lattice.

In order to address granularity we take the lattice defined by the supertypes \( t' \) of \( t_M \). To stay consistent with the use of join and meet above let the order be such that \( t' \leq t_M \) holds. In particular, \( t_M \) is the smallest type in that lattice. Now take a subset \( \mathcal{H}(M) \) of types in the lattice that defines a tree with root \( t_M \) – this will be called a set of hierarchical versions. In earlier versions of our work we actually requested a chain instead of a tree. Obviously, be means of canonical projections from \( \text{dom}(t_M) \) to \( \text{dom}(t) \) for each \( t \in \mathcal{H}(M) \), we obtain for each value \( v \) of type \( t_M \) and each \( t \in \mathcal{H}(M) \) a version \( v_t \) of type \( t \). Furthermore, due to the fact that we use a tree, we also get canonical up and down navigation operations between these versions. As \( t_M \) itself represents very condensed information, the hierarchical versions \( v_t \) represent lighter versions. Which version is the preferred one to start with depends on the preferences of the user. We should remark that there is no need to define specific style options for the hierarchical versions, as these carry over by means of the meet operation in the type lattice. In the same way, other OLAP-like operations, e.g. slicing and dicing are already defined by the content type expression, and there is no need to specify them.

With respect to adaptivity, we start from restrictions on the amount of data that is to be presented to a user. A value \( v \) of type \( t_M \) corresponds to the full amount of data represented by the media type \( M \), but this may not be needed or requested by a user, or considered to be not suitable due to network or end-device restrictions, e.g. restricted band-width, limited presentation capacity in videotext or simply user preference. The extension to interaction types dealing with these restrictions is cohesion, which permits a controlled splitting of a value \( v \) of type \( t_M \) into a sequence of values \( v_1, \ldots, v_\ell \) of types \( t_1, \ldots, t_\ell \) such that

\[
\bigcup_{i=1}^{\ell} t_i = t_M, \quad v_i \text{ is the canonical projection of } v, \text{ i.e. the decomposition is lossless, each } v_i \text{ contains a reference to } v_{i+1}, \text{ and each } v_i \text{ is a higher importance than } v_{i+1}.
\]

As discussed in [15] this can be achieved in basically two different ways, either by means of a cohesion pre-order, which extends the order defined by the type lattice to a total pre-order, or by means of proximity values for each pair of types in a specified maximal antichain in the type lattice.

The main difference between the two approaches to cohesion is that proximity values use an a priori decomposition of \( t_M \), i.e. an anti-chain of types \( t_1, \ldots, t_k \) such that \( \bigcup_{i=1}^{k} t_i = t_M \). Then the types in the cohesion sequence are either these a priori given types or result from applying the join operator to some of them. The guideline for defining the types in the cohesion anti-chain is that the joint type must be maximal within the limitations defined by the end-device, channel
or user preferences, and the sum of proximity values must be maximal. In this it is guaranteed that those \( t_1 \) with the highest cohesion will stay together.

The alternative of using a cohesion pre-order does not assume an a priori decomposition of \( t_M \). Then the guideline for selecting \( t_1 \) and thus define the value \( v_1 \) is to be maximal within the limitations defined by the end-device, channel or user preference and also maximal with respect to the cohesion pre-order. The process is then repeated recursively with a maximal type \( t' \) complementing \( t_1 \), i.e. \( t' \sqcup t_1 = t_M \).

Formal details of hierarchies and cohesion were presented in [15]. It is not too difficult to see that cohesion, hierarchies and style options are orthogonal extensions that can be mutually combined, i.e. for each hierarchical version the cohesion decomposition can be applied, and for each value in a cohesion sequence (for whatever hierarchical version) the style options can be applied.

4 Session and Context Support

As already sketched above media types enable interface abstraction non only on the level of elementary scenes, but can also be used for non-elementary scenes in the storyboard. In doing so media types can be used to support the concept of a session. Informally, a session comprises all scenes visited by a user from entering the systems until leaving it. Exploiting the hierarchies of scenes, a session can be represented itself by a non-elementary scene. That is, entering a session refers to the creation of the corresponding media object, which is linked to the elementary media objects used for defining subscenes. Technically, we model this by links from the media types associated with the elementary scenes to the overarching session scene. Such links can be modelled as being hidden, i.e. they actually do not appear as part of the navigation structure. The session object may represent data that is carried around through the session. A typical example is the shopping cart in e-commerce systems. It is deleted by means of garbage collection, i.e. when no more media object exists that links to it.

Another use of media types for complex scenes is the support of navigation context, for which the basic idea from contextual information bases has been adopted [18]. According to this a context is a set of object identifiers each having several names, and each of these names may be coupled with a reference to another context. There may be names for objects that are not referencing to other contexts. More formally, a context \( C \) is a finite set of triples \((o, n, r)\), where \( o \) is an object identifier, i.e. a value of some base type \( ID \), \( n \) is a name, i.e. a value of type \( STRING \), and \( r \) is either a reference \( \rightarrow C' \) to a context \( C' \) or nil, the latter one indicating that there is no such reference. We write \( C = \{n_1 : o_1 \rightarrow C_1, \ldots, n_\ell : o_\ell \rightarrow C_\ell\} \). If there is no reference for the \( i \)th name, i.e. we have \((o_i, n_i, nil)\) we simply omit \( \rightarrow C_i \) and write \( n_i : o_i \).

The idea of working with contextual information bases is that a user queries them and thus retrieves objects. In order to describe these objects in more details s/he accesses the context(s) of the object, which will lead to other objects by
following the references. In addition, a particular information encoded by the name is associated with each of these references.

Bringing together media types and contextual information bases, the obvious questions are: What are the object identifiers that are required in contextual information bases, if we are given media types? What are the references that are required in contextual information bases? Is it sufficient to have names for describing objects in a context or should these be replaced by something else?

The natural answer to the first question for generalising the notion of object identifiers in contexts is to choose the unique URLs of the media objects. Supporting navigation context requires access to path information, so we may want to reference back to the various media objects that we have encountered so far. These media objects are placed in several contexts, one of which is the right one corresponding to our path. However, we may also have different references, which lead to different contexts. So, the contexts we asked for in the second question are just the contexts for the media objects, which themselves are represented by media objects for paths, i.e. again non-elementary scenes.

As to the third question, we definitely want to have more information than just a name. Fortunately, the concept of media type is already based on the assumption of an underlying type system. Thus, we simply have to replace the names by values of any type permitted by the type system. Having defined such extended contexts, we can exploit query macros to traverse back a path in the story board and to explore alternative access paths, respectively.

However, one important aspect of media types is the use of classification abstraction. Conceptually, we do not define a set of media objects, but we generate them via queries defined on some underlying database schema. Therefore, we also need a conceptual abstraction for contexts. In order to obtain this conceptual abstraction, we assume another base type `Context`, the values of which are context names. Instead of this, we could take the type `URL`, but in order to avoid confusion we use a new type.

Then a `context` consists of a name `C`, i.e. a value of type `content`, a type `t_C` and a defining query `q_C`, which must be defined on the media schema, i.e. the set of media types, such that

\[
\{(\text{id} : \text{URL}, \text{value} : t_C, \text{reference} : \text{Context}), q_C\}
\]

defines a view. Thus, executing the query `q_C` will result in a set of triples \((u, v, r)\), where `u` is the URL of a media type, `v` is a value of type `t_C`, and `r` is the name of a context. If this context is undefined, this is interpreted as no reference for this object in this context.

5 Collaboration

In [1] we started an investigation of collaboration frameworks for distributed WISs addressing the problem that we may have tasks that are to be solved by collaboration of several users, e.g. using group work in e-learning systems.
Collaboration is understood as combining cooperation, communication and coordination. The ground idea for capturing collaboration in WISs therefore is to define another extension of media types by means of exchange frames, which combine an exchange architecture addressing communication and cooperation, and a coordination specification, which consists of supporting programs, cooperation style coordination facilities, roles of collaborators, their responsibilities and rights, and the protocols they rely on.

Exploiting again the idea of media types supporting complex scenes we can abstract from the actor representing an individual user to actors representing a group of users. The exchange architecture for cooperation and communication within a group can be modelled by means of a workspace, which again is nothing but a media object associated with the group action, and consequently media object used by individuals in the group contain hidden links to the group object.

6 Reasoning about Media Types

Following our previous work in [12] we can formalise the operations associated on media types using the following language of abstract programs:

- 1 and 0 are abstract programs meaning skip and fail, respectively.
- An assignment \( x := \text{exp} \) with a variable \( x \) and an expression of the same type as \( x \) is an abstract program. The possible expressions are defined by the type system. In addition, we permit expressions \( \{ \mathcal{P} \} \) with a logic program \( \mathcal{P} \), assuming that \( \mathcal{P} \) contains a variable \( \text{ans} \). The expression \( \{ \mathcal{P} \} \) is interpreted as the result of the logic program bound to \( \text{ans} \).
- If \( p, p_1 \) and \( p_2 \) are abstract programs, the same holds for the iteration \( p^* \), the choice \( p_1 + p_2 \) and the sequence \( p_1 \cdot p_2 = p_1 p_2 \).
- If \( p \) is an abstract program and \( \varphi \) is a condition, then the guarded program \( \varphi p \) and the postguarded program \( p \varphi \) are also abstract programs.
- If \( x \) is a variable and \( p \) is an abstract program, then the selection \( @x \cdot p \) is also an abstract program.

On these grounds we introduce a higher-order dynamic logic, where the order comes from the intrinsic use of the set constructor and the logic programs in queries. In fact, instead of using logic programs with a semantics defined by fixed-points, we could use directly higher-order logic enriched with a fixed-point operator. As a consequence, we may consider a logic program \( \mathcal{P} \) as a representative of a higher-order logical formula, say \( \varphi_{\mathcal{P}} \). If \( \{ \mathcal{P} \} \) is used as the right-hand side of an assignment, then it will correspond to a term \( \text{I\text{ans}}.\varphi_{\mathcal{P}} \) denoting the unique \( \text{ans} \) satisfying formula \( \varphi_{\mathcal{P}} \). That is, all conditions turn out to be formulae of a logic \( \mathcal{L} \), which happens to be a higher-order logic with a fixed-point operator. From the point of view of expressiveness the fixed-point operator is already subsumed by the order, but for convenience we do not emphasise this aspect here. Furthermore, by adding terms of the form \( \text{I}x.\varphi \) with a formula \( \varphi \) and a variable \( x \) all assignments in operations are just “normal” assignments, where the left-hand side is a variable and the right-hand side is a term of \( \mathcal{L} \).
We now extend $\mathcal{L}$ to a dynamic logic by adding formulae of the form $[p]\varphi$ with an abstract program $p$ and a formula $\varphi$ of $\mathcal{L}$. Informally, $[p]\varphi$ means that after the successful execution of $p$ the formula $\varphi$ necessarily holds. In addition, we use the shortcut $\langle p \rangle \varphi \equiv \neg [p] \neg \varphi$, so $\langle p \rangle \varphi$ means that after the successful execution of $p$ it is possible that the formula $\varphi$ holds. Using our recursive definition of abstract programs the following rules apply to $[p] \psi$ for a complex abstract program $p$:

\[ [1] \psi \equiv \psi \]
\[ [0] \psi \equiv 0 \]
\[ [x := t] \psi \equiv \psi\{x/t\} \] (substitute all free occurrences of $x$ in $\psi$ by $t$)
\[ [p_1 p_2] \psi \equiv [p_1] [p_2] \psi \]
\[ [p_1 + p_2] \psi \equiv [p_1] \psi \land [p_2] \psi \]
\[ [p^*] \psi \equiv \text{the weakest solution } \varphi \text{ of } \varphi \leftrightarrow \psi \land [p] \varphi \]
\[ [\varphi p] \psi \equiv \varphi \rightarrow [p] \psi \]
\[ [p \varphi] \psi \equiv [p] (\varphi \rightarrow \psi) \]
\[ [\exists x \cdot p] \psi \equiv \forall x. [p] \psi \]

As outlined in [12] we can formulate proof obligations for the operations that result from the specification of the story space, in particular for consistency with respect to static and dynamic integrity constraints on the underlying database schema. Furthermore, we can extend WIS personalisation in the light of dynamic logic.

7 Conclusion

In this survey we outlined in compact form the potential of the concept of media type emphasizing its use for interface abstraction supporting adaptivity to users, channels and devices, collaboration among user groups, session and context support, and serving as the conceptual bridge between identified requirements, the usage model of the WIS, the layout and playout, and the implementation on the basis of XML clusters and scripts. The versatility of media types are a major reason for the success of the codesign approach to WIS design as demonstrated in various large-scale applications. Furthermore, it is worth exploring the concept of media types in more depth, in particular with respect to the logical inferences as started in [12] on the basis of dynamic logic, the connection to deontic constraints on the storyboard and their refinement to the level of media types [13], and their embedding in layout and playout by means of screenography [10].

References


Engineering Database Component Ware

Bernhard Thalheim

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

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

1 Towards Information Systems Engineering

Component-Based Application Engineering.

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

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

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

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

![Fig. 1. HERM Representation of the Star Type Screw](image)

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

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

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

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

1 We use the extended ER model [Tha00] that allows to display subtypes on the basis of unary relationship types and thus simplifies representation.
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 modelling such as modular design by units [Tha00] are used. It is an abstraction technique based on principles of hiding and encapsulation. Design by units allows to consider parts of the schema in a separate fashion. The parts are connected via types which function similar to bridges.

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

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

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

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

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

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

**Goals of the Paper.**

The paper surveys our approach [Tha02, Tha03a, Tha05] for systematic development of
large database schemata and applies it for database construction based on components and for collaborating component suites. The paper is based on [Fey03, FT02, ST06a, ST04].

We introduce first the concept of database components and then discuss engineering of database applications based on components.

## 2 Database Components and Construction of Schemes

Database Schemes in a Nutshell.

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

A **database type** $\mathcal{S} = (S, O, \Sigma)$ is given by

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

Objects of the database type $\mathcal{S}^C$ are $S$-structured. Classes $\mathcal{S}^C$ 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 $\mathcal{S}^C$ and “modification” operations allowing to change the objects in the class $\mathcal{S}^C$ if static and dynamic integrity constraints are not invalidated.

A **database schema** $\mathcal{D} = (\mathcal{S}_1, \ldots, \mathcal{S}_m, \Sigma_G)$ 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 $\mathcal{V} = (V, O_V)$ on a schema $\mathcal{D}$ via

- an (parameterized) algebraic expression $V$ on $\mathcal{D}$ and
- a set of (parameterized) operations of the HERM algebra applicable to $V$.

The view operations may be classified too into retrieval operations $O^R_V$ and modification operations $O^M_V$. Based on this classification we derive an output view $O^V$ of $\mathcal{V}$ 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 [Tha00].
Database Components and Component Algebra.

A database component is database scheme that has an import and an export interface for connecting it to other components by standardized interface techniques. Components are defined in a data warehouse setting. They consist of input elements, output elements and have a database structuring. Components may be considered as input-output machines that are extended by the set of all states $S^C$ of the database with a set of corresponding input views $I^V$ and a set of corresponding output views $O^V$. 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 $K = (S_K, I^V_K, O^V_K, S^C_K, \Delta_K)$ is specified by

- **static schema** $S_K$ describing the database schema of $K$,
- **syntactic interface** providing names (structures, functions) with parameters and database structure for $S^C_K$ and $I^V_K, O^V_K$,
- **behavior** relating the $I^V, O^V$ (view) channels
  \[ \Delta_K : (S^C_K \times (I^V_K \rightarrow M^*)) \rightarrow P(S^C_K \times (O^V_K \rightarrow M^*)) \].

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

Components $K_1 = (S_1, I^V_1, O^V_1, S^C_1, \Delta_1)$ and $K_2 = (S_2, I^V_2, O^V_2, S^C_2, \Delta_2)$ are free of name conflicts if the set of attribute, entity and relationship type names are disjoint.

Channels $C_1$ and $C_2$ of components $K_1 = (S_1, I^V_1, O^V_1, S^C_1, \Delta_1)$ and $K_2 = (S_2, I^V_2, O^V_2, S^C_2, \Delta_2)$ are called domain-compatible if $\text{dom(type}(C_1)) = \text{dom(type}(C_2))$.

An output $O^V_1$ of the component $K_1$ is domain-compatible with an input $I^V_2$ of the component $K_2$ if $\text{dom(type}(O^V_1)) \subseteq \text{dom(type}(I^V_2))$.

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

---

**Fig. 2. The Composition of Database Components**

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

The star schema is the main component schema used for construction. A star schema for a database type \( C_0 \) is defined by:

- the (full) (HERM) schema \( S = (C_0, C_1, ..., C_n) \) covering all types on which \( C_0 \) has been defined,
- the subset of strong types \( C_1, ..., C_k \) forming a set of keys \( K_1, ..., K_s \) for \( C_0 \), i.e., \( \bigcup_{i=1}^{s} K_i = \{C_1, ..., C_k\} \) and \( K_i \to C_0 \), \( C_0 \to K_i \) 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 \( \text{card}(C_0, C_i) = (0, 1) \) 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 specialisations forming a specialization tree \( C_{l+1}, ..., C_n \).
- Representation based on a relationship type \( C_0 \) with components \( C_1, ..., C_k \), with attributes \( C_{k+1}, ..., C_l \) and specialisations forming a specialization tree \( C_{l+1}, ..., C_n \).

In this case, \( C_0 \) is a pivot element [BP00] in the schema.

- Representation by be based on a hybrid form combining the two above.

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

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

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

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

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

Lifespan variants of objects and their properties may be explicitly stored. The lifespan of products in the acquisition process can be based on the Product-Quote-Request-Response-Requisition-Order-InventoryItem-StoredItem cycle displayed in Figure 6.

Meta-Characterization of Components, Units, and Associations.

Utilization information is often only kept in log files. Log files are inappropriate if the utilization or historic information must be kept after the data have been changed.
Database applications are often keeping track of utilization information based on archives. The same observation can be made for schema evolution. We observed that database schemata change already within the first year of database system exploitation. In this case, the schema information must be kept as well.

The skeleton information is kept by a meta-characterization information that allows to keep track on the purpose and the usage of the components, units, and associations. Meta-characterization can be specified on the basis of docket information. The following frames follows the co-design approach with the integrated design of structuring, functionality, interactivity and context. The frame is structured into general information provided by the header, application characterization, the content of the unit and documentation of the implementation.

– on the content (abstracts or summaries),
– on the delivery instruction,
– on the parameters of functions for treatment of the unit (opening without zooming, breath, size, activation modus for multimedia components etc.)
– on the tight association to other units (versions, releases etc.),
– on the meta-information such as resources, restriction, copyright, roles, distribution policy etc.
– on the content providers, content reviewers and review evaluators with quality control policies,
– on applicable workflows and the current status of completion and
– on the log information that enable in tracing the object’s life cycle.

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

3 Non-Invasive Database Component Composition

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

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

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

Given a sets of components \( \mathcal{R} = \{K_1, \ldots, K_m\} \) and labels \( \mathcal{L} = \{L_1, \ldots, L_n\} \) with \( n \geq m \). Given furthermore a total function \( \tau : \mathcal{L} \rightarrow \mathcal{R} \) used for assigning roles to
components in harnesses. The triple \((K, L, \tau)\) is called **harness skeleton** \(S\). 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.

![Diagram](image)

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

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

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

Filters may be represented either graphically or in a tabular form. In our example, we obtain the following filter. Components are already presented in Figure 3. We de-
velop a number of services which might be used depending on the role, rights, and positions of the users. For instance, the parliamentarian is interested in search of related documents in the role of an inhabitant and in search of related meetings.

The implementation of filters is rather straightforward. Each harness has a filter. Since views are defined together with their identification mechanism, an n-ary harness may be represented by an \( (n+1) \)-ary relationship type associating the components with their roles and extended by the filter.

A harness consists of the harness skeleton \( h = (K, L, \tau) \) and the harness filter \( \delta = \{(L_i, V^{L_i}) | 1 \leq i \leq n, L_i \in L, V^{L_i} \subseteq V_{\tau(L_i)}\} \) for a set of wrapped components \((K_i, V_i)\).

Operators Used For Non-Invasive Schema Construction.

In [Tha03b] a number of composition operators for construction of entity and relationship types has been introduced: constructor-based composition, bulk composition, lifespan composition (architecture-based composition, evolution composition, circulation composition, incremental composition, network composition, loop composition), and context composition.

We generalize now these composition operators to component-based schema construction.

**Constructor harnesses** are based on composition operations such as product, nest, disjoint union, difference and set operators.

**Bulk harnesses** allow to bound components, types or classes which share the same skeleton. Two harness skeletons \( h_1 = (K_1, L_1, \tau_1) \) and \( h_2 = (K_2, L_2, \tau_2) \) are called
unifiable if they are defined over the same set of components, \(|\Sigma_1| = |\Sigma_i| = n_i\), and there exists a permutation \(\rho\) on \(\{1, \ldots, n\}\) such that \(K_{\tau_1(i)} = K_{\tau_2(\rho(i))}\). The bulk harness of unifiable harnesses \(H_1, \ldots, H_p\) is constructed by renaming the labels \(L_j\) of each harness \(H_j\) 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** displayed in Figure 4.

![Fig. 4. The Waffle Architecture Composition](image)

An Application of Component Composition.

A typical lifespan construction is the **Order** chain displayed in Figure 6. We discover a chain in the ordering and trading process: **Quote**, **Request**, **Response**, **Requisition**, **Order**, **Delivery**, **Billing**, **Payment**. Within this chain, parameters such as people responsible in certain stages are inherited through the components. They are included into the type for the purpose of simpler maintenance. They cannot be changed within the type inheriting the component. Thus, we use an extended inheritance of structuring beyond the inheritance of identification.

At the same time, this schema can be constructed on the basis of components. We may distinguish only four basic parts. Parties are either organisations or people. Products have a number of properties that are independent on parties. The two components are associated within the ordering and trading process. The parties may play different roles within this process. The parties act based on these roles. So, the component schema is given in Figure 5.

The roles of parties in the ordering and trading process can be unfolded. We observe a role of a supplier, of a requestor, of a responding party, of a requisition party and
finally the role of the orderer. At the same time, the final order has a history or a lifespan. We may apply the lifespan constructor as well. The application can be either based on collaborating components are can be condensed to the schema given in Figure 6. This schema combines components and unfolds roles and expands the ordering and trading activities. We notice that this schema is not necessarily the solution for the ordering and trading process. We may use the components instead and explicitly model component collaboration. In this case the components may stay non-integrated.

4 Collaborating Database Component Suites

Services Provided By Components For Loosely Coupled Suites. A service consists of a wrapped component \((K_i, V_i)\), the competencies \(\Sigma_{(K_i, V_i)}\) provided and properties \(\Psi_{(K_i, V_i)}\) guaranteeing service quality. Wrapped components offer their own data and functions through their views. The competence of a service manifests itself in the set of tasks \(T\) that may be performed and in the guarantees for their quality.
Database Component Collaboration.
Instead of expanding and unfolding the component schema in Figure 5 we may follow a different paradigm. The four basic parts are loosely associated by a collaboration, are supported by component databases and communicate for task resolution. This approach has already been tried for distributed databases. Our approach is far more general and provides a satisfying solution.

A collaborating database component suite \( \Theta = (\mathcal{H}, \mathfrak{F}, \Sigma) \) consists of
- an set \( \mathcal{H} \) of wrapped database components \((K_i, V_i)\)
- a harness consisting of the harness skeleton \(\mathcal{H} = (\mathcal{K}, \mathcal{L}, \tau)\) and the harness filter \(\mathfrak{F}\),
- an collaboration schema \(\mathfrak{F}\) among these components based on the harness, and
- obligations \(\Sigma\) requiring maintenance of the collaboration.

The collaboration schema explicitly models collaboration among components. We distinguish three basic processes of component collaboration:

Communication is defined via exchange of messages and information or simply defined via services and protocols [Kön03]. It depends on the choice of media, transmission modes, meta-information, conversation structure and paths, and on the restriction policy. Communication must be based on harnesses.

Coordination is specified via management of components, their activities and resources. It rules collaboration. The specification is based on the pre-/post-articulation of tasks and on the description management of tasks, objects, and time. Coordination may be based on loosely or tightly integrated activities, may be enabled, forced, or blocked. Coordination is often specified through contracts and refines coordination policies.

Cooperation is the production of work products taking place on a shared space. It can be considered as the workflow or life case perspective. We may use a specification based on storyboard-based interaction that is mapped to (generic and structured) workflows. The information exchange is based on component services [ST06a] for production, manipulation, organization of contributions.

This understanding has become now a folklore model for collaboration but has not yet been defined in an explicit form. We use the separation of concern for the specification of component collaboration.

Collaboration obligations are specified through the collaboration style and the collaboration pattern.

The collaboration style is based on four components describing supporting programs of the connected component including collaboration management;

- data access pattern for data release through the net, e.g., broadcast or P2P, for sharing of resources either based on transaction, consensus, and recovery models or based on replication with fault management, and for remote access including scheduling of access;
- the style of collaboration on the basis of component models which restrict possible communication;
- and the coordination workflows describing the interplay among parties, discourse types, name space mappings, and rules for collaboration.
Collaboration pattern generalize protocols and their specification [Kön03]. They include the description of components, their responsibilities, roles and rights. We know a number of collaboration pattern supporting access and configuration (wrapper, facade, component configuration, interceptor, extension interface), event processing (reactor, proactor, asynchronous completion token, accept connector), synchronization (scoped locking, strategized locking, thread-safe interface, double-checked locking optimization) and parallel execution (active object, monitor object, half-sync/half-async, leader/followers, thread-specific storage).

Supporting Collaboration Schemata By Service Managers. The abstraction layer model [Tha00,ST06b] distinguishes between the application domain description, the requirements prescription, the system specification, and the logical or physical coding. The specification layer typically uses schemata for specification. These schemata may be mapped to logical codings. The mapping of services to logical database components is already given by classical database textbooks. We map collaboration schemata to service managers. This mapping provides also a framework for characterisation of competencies and quality.

The service manager $Man$ supports functionality and quality of services and manages sets of wrapped components. The manager supports a number of features for collaboration. The architecture of the services manager follow the separation of concern into communication, coordination, and cooperation. We may thus envision the architecture in Figure 7.

**Fig. 7.** Layers of a services manager for typical collaborating components

Collaborating services are defined by the quadruple $S = (\Sigma, Man, \Sigma_S, \Psi_S)$ describing (Collaborating Suite, Service Manager, Competence, Characteristics). The competence is derived from the competence of the services. The quality of collaborating services may also be derived from the quality properties of components in the suite based on the properties of the harnesses, their collaboration schema, and the corresponding obligations. Typically, quality heavily depends on the suite properties. For instance, reliability of a suite may be less than the reliability of its components.
Concluding by Demonstrating the Potential of Privacy Supporting Suites. Let us show the potential of loosely coupled database component suites for privacy workbenches. Privacy research is becoming the “poor cousin” among the mainstream research. Novel applications such as Web2.0 have created a new rush towards social networking and collaborative applications. This enables new possibilities, but also is a threat to users’ privacy and data. On the surface, many people seem to like giving away their data to others in exchange for building communities or like to get bribes from companies in exchange of privacy. A number of hidden privacy implications of some Web2.0 and Identity2.0 services, standards and applications can be observed here. At the same time, it is often stated that there is no way to properly preserve privacy.

We show the potential of collaborating databases based on the infon model of [AFFT05]. An infon is a discrete item of information of an individual and may be parametric. The parameters are objects, and the so-called anchors assign these objects such as agents to parameters.

We may distinguish four relationships between infons and individuals (people), institutions, agencies, or companies: An infon may be possessed by an individual, institution, agency, or company. For example, an individual may possess private information of another individual or, a company may have in its database, private information of someone. Individuals know that an infon is in possession of somebody else. Infons may belong to individuals. Finally, an infon is owned by an individual. The ownership is the basis for the specification of privacy.

The owner sovereignty principle restrains the right or sovereignty of people over their owned infons. A policy supporting the owner sovereignty principle restrains the possessor in the role of ‘content and topic observer’ and preserves the owner in the role of ‘informed owner’ and ‘refresher’. The contract between owner and possessor restricts the possibilities and rights of the possessor for using content and topics on an ongoing basis by additional actions such as

- to monitor activities of the possessor,
- to collect information (about conditions of possession),
- to give a warning to the owner, and
- to take actions such as use, security, welfare, accuracy, correctness, and maintenance of infons to the owner.

The collaboration is faithful if the portfolio and profile of contracting possessor do not include any forbidden action or ability, all reporting obligations are observed, and the proprietor is able to observe obligations applied to the possessor.

The private database is called information wallet if it is a component service with the following additional function enhancements for owners o, possessors p, infons i, infon requests r, time stamps t, delivered infon streams identifiers s, public keys puk(r, o, p, t) for p, private keys prik(i, o, p, t) for o, records of delivered infons by the owner store(o, i, p, s), and encoding and decoding relations encrypt(i, prik, s) for o, decrypt(p, r, s, puk, t) extended by steganographic watermarking mark(i, o, p) for infons:

- satisfy(request(r, o, p, t)) ⇒ encrypt(i, prik(i, o, p, t), s) ∧ deliver(p, o, s) ∧ store(o, i, p, s)
- decrypt(p, r, s, puk(s, o, p, t), t') ⇒ inform(o, Act(p, s, decrypt), t') ∧ mark(i, o, p)
\( \text{read}(p, \text{mark}(i, o, p), t') \Rightarrow \text{inform}(o, \text{Act}(p, s_i, \text{read}), t') \)

\( \text{send}(p, \text{mark}(i, o, p), p', t') \Rightarrow \text{inform}(o, \text{Act}(p, s_i, \text{send}(p, p')), t') \land
\neg \text{send}(p, \text{mark}(i, o, p), p', t') \land \text{send}(p, r, p', t') \)

\( \text{satisfy}(\text{request}(\text{puk}(s, o, p, t)), o, p, t') \Rightarrow
\text{deliver}(p, o, \text{puk}(s, o, p, t)) \land \text{store}(o, i, p, \text{puk}(s, o, p, t)) . \)

We assume that watermarked infons cannot be changed by anybody. We can show now that information wallets preserve the owner sovereignty principle.

\textbf{References}


Semantics in Data and Knowledge Bases

Klaus-Dieter Schewe\textsuperscript{1} and Bernhard Thalheim\textsuperscript{2}

\textsuperscript{1} Information Science Research Centre
20A Manapouri Cr, Palmerston North 4410, New Zealand
\textsuperscript{2} Christian Albrechts University Kiel, Department of Computer Science
Olschnausenstr. 40, D-24098 Kiel, Germany
kdschewe@acm.org thalheim@is.informatik.uni-kiel.de

1 Semantics

Semantics is the study of meaning, i.e. how meaning is constructed, interpreted, clarified, obscured, illustrated, simplified, negotiated, contradicted and paraphrased [Wan87]. It has been treated differently in the scientific community, e.g., in the area of knowledge bases and by database users.

– The scientific community prefers the treatment of ‘always valid’ semantics based on the mathematical logic. A constraint is valid if this is the case in any correct database.
– Database modellers often use a ‘strong’ semantics for several classes of constraints. Cardinality constraints are based on the requirement that databases exist for both cases, for the minimal and for the maximal case.
– Database mining is based on a ‘may be valid’ semantics. A constraint is considered to be a candidate for a valid formula.
– Users usually use a weak ‘in most cases valid’ semantics. They consider a constraint to be valid if this is the usual case.
– Different groups of users use an ‘epistemic’ semantics. For each of the group its set of constraints is valid in their data. Different sets of constraints can even contradict.

Semantics is currently one of the most overused notions in modern computer science literature. Its understanding spans from synonyms for structuring or synonyms for structuring on the basis of words to precise defined semantics. This partial misuse results in a mismatch of languages, in neglecting formal foundations, and in brute-force definitions of the meaning of syntactic constructions.

1.1 Variety of Notions for Semantics

We are thus interested in a clarification of the notion of semantics. The notion of semantics is used in different meanings:

\textbf{Lexical semantics} is the study of how and what the words in a language denote. It uses a theory of classification and decomposition of word meaning and of association of words via relationships such as hyponomy, synonymy, troponymy, hyponymy, and antonymy. Word fields are the main notion for lexical semantics. An ontology is
Semantics in Data and Knowledge Bases

typically based on word fields in a rudimentary form and on selected associations among words. In general, an ontology provides a shared and common understanding of a domain that can be communicated between people and heterogeneous and distributed application systems. This meaning is the basis for “Semantic Web. It uses a rudimentary form of word semantics for meta-characterisation.

**Grammatical semantics** uses categories such as ‘noun’ for things, ‘verb’ for actions and ‘adjectives’ for descriptions. Categories are enhanced by grammatical elements. Grammatical rules describe which syntactic expressions are well-formed. It is often assumed that the syntactic properties of words are determined by their meaning. Combinatorial semantics is a specific form of grammatical semantics.

**Statistical semantics** bases the meaning on co-occurrence of words, on pattern of words in phrases, and on frequency and order of recurrence. Information retrieval applications use this co-occurrence pattern for similarity detection, keyword extraction, measuring cohesiveness in texts, discovering different senses of the word, and for analysing and mining.

**Logical semantics** is based on a relation between the formal language of logics defined on some alphabet or signature from one side and the set of structures or worlds defined on the same signature from the other side. The relation serves as the means for realising the assignment and characterises at the same time the truth of expressions via their interpretation in the set of structures. The truth value of an expression is a statement at a meta-language level.

**Prototype semantics** bases meaning on users evolving experience. It is thus culture and community dependent. The semantical structure used for definition of the meaning consists of different elements which have unequal status. This leads to a graded notion and varying structuring within semantical structures. Some elements are better for clarification of the concepts. Elements are mainly possible contributors to the meaning. Semantical structures may be layered into basic level ones and combined ones.

**Program and dynamic semantics** is based on the semantic memory, i.e. the memory of meanings, understandings, and other concept-based knowledge unrelated to specific experiences of agents. Dynamic semantic structures may be represented by generic [BST06] or abstract structures (e.g. semantic networks, associative or feature or statistical models) that apply to a wide variety of experimental objects. It overcomes limitations of the nativist view on semantics and explicitly considers evolution of meaning depending on a state of a reference system such as the computer and depending on the context and agents. Database semantics is based on expression-linear definition of meaning and allows to reduce the meaning to those components that are necessary for definition of the meaning. Dynamic semantics supports semantic under-specification and transfer of meaning by injection of context and agent opinion.

Semantics can be based

- on constructors and compositions in the sense of Frege that the interpretation can be defined by the interpretation of a constructor and an interpretation of constituents of complex expressions and the interpretation of basic is provided by assumptions,
– on context in the sense of reduction logics by assigning a set of rules for expansion of expressions to expressions for a meaning is defined and by defining agents with roles such as hearer and speaker that provide an understanding in their worlds, e.g., reduction semantics used in database semantics for reduction of components of sentences to proplets, and
– on transformations to other languages for which a semantics is well-defined and by transformation rules which meaning is well-understood and and concise.

The principle of compositionality postulates that the meaning of an expression is determined by its structure and its constituents. This principle is productive in the sense that meaning can be derived without any explicit definition in advance and is systematical in the sense that there are definite and predictable pattern among expressions. Expressions are referentially transparent since they can be replaced or instantiated by values without changing the expression. Queries are referentially transparent whereas commands in non-functional languages may change the environment and are context-dependent. Compositionality is one of the main conditions for axiomatisability of logics. Compositionality cannot be maintained within context-dependent or time-dependent structures.

Considering more complex type systems the treatment of missing values adds another dimension to semantics. For instance, in the entity-relationship model [Tha00a] entity types can be defined on set semantics and relationship types can be based on pointer (or reference) semantics enabling thus the developer to reason on ‘missing links’. This difference in the treatment of semantics has led to some of the problems discussed for view integration. We therefore established this workshop seria together with the FoIKS conferences in order to maintain the knowledge on semantics in both fields.

1.2 The Semiotics Background of Semantics and Pragmatism

Database and knowledge base theory use languages. Therefore, we are bound to the conceptions of the language, the expressivity of the language, and the methodology for language utilisation. The Sapir-Whorf observation [Who80] postulates that developers skilled in a certain language may not have a (deep) understanding of some concepts of other languages.

Semantics is the study and knowledge of meaning in languages. It is a central component of semiotics. Modern semantics [Hau01] is based on a notion of meaning and reference, on a notion of semantic spaces, on a construction of semantic structures such as semantic fields, on a notion of semantic relations among constructions, on semantic components, and on a notion of prosodic, grammatical, pragmatical, social and propositional meaning of constructions. Semiotics distinguishes between syntactics (concerned with the syntax, i.e., the construction of the language), semantics (concerned with the interpretation of the words of the language), and pragmatics (concerned with the meaning of words to the user). Most languages used in computer science have a well-defined syntax. Their semantics is provided by implementations which are often not made public and their pragmatics is left to experimentation by the user. This ‘banana’ or ‘potato’ principle leads to very different utilization and does not allow consistent use by groups of developers and over time. Each step is based on a specification language that has its syntax, semantics, and pragmatics.
Syntactics: Inductive specification of structures uses a set of base types or a vocabulary or an alphabet, a collection of constructors and an theory of construction limiting the application of constructors by rules or by formulas in deontic logics. In most cases, the theory may be dismissed. Structural recursion is the main specification vehicle. Constructors may be defined on the basis of grammatical rules.

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

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

These main branches of semiotics cannot be entirely layered and are intertwined. The meaning of a word depends on its use [Wit58] and its users. We often assume however that syntax is given first. We should distinguish between expression meaning, statement meaning, and context or utterance meaning. The first one designates the semantic properties an expression possesses merely by the virtue of being well formed and is based on the propositional content of the expression, i.e. propositions or semantical units it contains, on the interrogative, imperative, and/or expressive meaning, and finally on features of formal distinction. A statement postulates that some state of affairs holds. It has a truth value. The statement meaning has two parts: an element of assertion and something that is asserted. What is asserted is called proposition. The proposition is an expression that can be believed, doubted, or denied or is either true or false. The simplest type of proposition consists of an argument and a predicate. The context meaning depends on the intention of the sender and the relationship of the sender with the speaker and is based on auxiliary expressions shared between sender and receiver.

We also may distinguish between the existential approach to meaning based on a correlation of expressions in a language with aspects in the world. The intentional approach associates some kind of representation with concepts as the main constituents of the sense and depends on the cultural context. Semantical units or propositions are interrelated by entailment. Entailment is different from material implication and relates propositions by forward propagation of truth and backward propagation of falsity. Propositions can be contraries, contradictories, or independent. They may belong to a category or genre of expression, are given in a certain style or manner, are often based on stereotypical norms of expression, depend on ideas and values that are employed to justify, support or guide the expression, reflect aspects of culture or social order, are shaped according to the community that uses them, and are configured by theories or paradigms.

Pragmatism means a practical approach to problems or affairs. According to Webster [Web91] pragmatism is a ‘balance between principles and practical usage’. Thus, it is a way of considering things. Pragmatism may be based on methodologies, e.g., database design methodologies. For instance, the co-design methodology of database development [Tha00a] enables in consistent design of structuring, functionality, interactivity and distribution of information systems. The constituents of the methodology are,
however, constructs of the design language and constructors used to construct complex construction. Constructs we use are the elements of the extended entity-relationship models [Tha00a]: attribute types, entity types, relationship types of arbitrary order, and cluster types. Constructors used for defining more complex types are: (Cartesian) product constructor (for associating types) and list, tree set and bag constructors (for constructing groups or collections of types)[Tha00b]. The way of considering things or the practical usage has been investigated in less depth. Schemata have a certain internal structuring or meta-structuring. Thus, pragmatism of modelling should also be based on principle to handle the meta-structuring within the schemata.

1.3 Formal Semantics in General

Formal semantics is typically based on a system that allows to reason on properties of systems. It aims in describing syntactic and semantic types or elements that are used as well as the meaning function that maps syntactic types to semantic types. There are several forms of formal semantics: operational, denotational, axiomatic, transformational, algebraic, macro-expansion, and grammar semantics.

Formal semantics typically assume compositionality. The principle of compositionality is not necessary and may contradict applications. Compositionality assumes that the elements are independent from each other. For instance, the classical XOR-connective $\neg (\alpha \land \beta)$ defines incompatibility of $\alpha$ and $\beta$. It thus provides a context for subexpressions. Idioms, non compounds, active zones and complex categories cannot be based on compositionality. If compositionality is assumed we may distinguish between the additive mode where the meaning of components is added for the constructor and the interactive mode where the meaning of at least one components is radically modified. Formal semantics mainly uses the additive mode. Natural, lexical, prototype, statistical and partially program semantics use the interactive mode. The meaning of a complex expression can either be of the same basic type as one of the components (endocentric) or of a different type (exocentric).

Different variants of semantics is used in computer science [Bak95,BS03,Gun92],[Mos92,Rey99,Ten97,Tha00a]. A formal semantics [Cry87,Sch71,Sch72,Ste73] is typically given

- by an interpreter that maps syntactic types to semantic types,
- by a context abstraction that is based on an aggregation of values which remain fixed in certain temporal and spatial intervals,
- by states that provide a means of representing changes over time and space,
- by an configuration that is based on an association of contexts and states,
- by an interpretation function that yields state results based on certain computation,
- by an evaluation function that yield some value results for the syntactic types, and
- by an elaboration function that yield both state and some other value results.

These mapping are often given in a canonical setting, i.e. interpreters are defined on signatures, context abstraction is neglected due to use of the most general types, states are based on mathematical structures, configurations are fixed, and interpretation functions are given in a canonical setting. The evaluation function can be freely configured.
The elaboration function is used for the definition of aggregates such as the quantifiers. Mathematical logics and formal semantics in computer science is defined by such canonical setting.

In computer science we may distinguish between static semantics that lays down what it means for terms to be well defined (beyond syntactic criteria) and dynamic semantics that determines the meaning of terms. Dynamic semantics is mainly defined through operational semantics either via small-step operational semantics (structural operational semantics or reduction semantics) or big-step operational semantics (natural semantics or evaluation semantics), or through denotational semantics or through axiomatic semantics. Additionally, transformational and algebraic approaches are used.

These seven mappings are the basis for a variety of definitions of semantics. Formal semantics does not oblige any restriction by states or context. It uses a strict matching between the syntactical language and the world of semantical structures either by strict association or by embedding of expressions in the syntactical language \( L \) into the language \( L' \) in the world of semantical structures. In the opposite, matching is only partial for natural logics. We may for instance use different kinds of mappings for different logics:

<table>
<thead>
<tr>
<th>mappings</th>
<th>logics</th>
<th>closed world logics</th>
<th>logics on finite worlds</th>
<th>logics on natural worlds</th>
</tr>
</thead>
<tbody>
<tr>
<td>matching of syntactic language ( L ) and semantic structure worlds ( \forall ) on signature ( \tau )</td>
<td>exact or coincidence embedding</td>
<td>exact</td>
<td>exact or coincidence embedding</td>
<td>partial depending on interest and meaning in use</td>
</tr>
<tr>
<td>considering context</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>depending on use and user</td>
</tr>
<tr>
<td>considering states</td>
<td>any</td>
<td>any</td>
<td>only finite structures</td>
<td>states in scope</td>
</tr>
<tr>
<td>restricting states and context</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>depending on interest and demand</td>
</tr>
<tr>
<td>interpretation for alphabets</td>
<td>exact for alphabet</td>
<td>exact</td>
<td>potentially restricted</td>
<td>multiple interpretations</td>
</tr>
<tr>
<td>evaluation of variables</td>
<td>full</td>
<td>full</td>
<td>full</td>
<td>partial evaluation</td>
</tr>
<tr>
<td>elaboration</td>
<td>full</td>
<td>full</td>
<td>partial evaluation</td>
<td>derivation structures</td>
</tr>
</tbody>
</table>

This table shows the variety of approaches that might be used for the definition of semantics. Application models restrict languages to those parts that are currently of interest. The context also considers enterprise and social models. Integrity constraints used in database models are not interpreted for specific values such as null values. Interpretation of expressions is restricted to meaningful expressions in finite worlds, e.g. the tuple-relational calculus in relational database theory only defines the meaning of those expressions that map finite structures to finite structures.

Formal semantics may be understood as the semantics of formal languages. Model-theoretic semantics is widely used in computer science, e.g. for semantics of databases. It defines the meaning by recursive mapping to some predefined mathematical structures and by assigning propositions to truth values. Proof-theoretic semantics is widely used in artificial intelligence and for knowledge bases. It associates the meaning of propositions with the roles that they play in inferences. Truth-value semantics assigns a truth
value to expressions without reference to structures. Game and probabilistic semantics are other kinds of formal semantics.

1.4 Semantics of Mathematical Logics as One Kind of Semantics

Mathematical logics uses a canonical way of associating syntactic and semantic types. Additionally, the semantic type and the syntactic type have the same signature. The expressions of syntactic types are inductively constructed starting with some basic expressions of certain construct by application of expressions of some other construct. For instance, we may start with truth values and variables. Terms and formulas are then based on these basic expressions.

The semantic type \( LOG \) of first-order mathematical logics uses the two truth values \( true \) and \( false \), canonical connectives \( \land, \lor \) and \( \lnot \) and the quantifiers \( \forall \) and \( \exists \).

The interpreter is given by a canonical mapping. Context abstraction is defined by the trivial context and thus not considered. Most logics do not consider changing states. The interpretation function is an inductive function that is based on interpretation of basic expressions and follows the construction of expressions. The evaluation and elaboration functions allow to consider different value assignments to an expression.

The correspondence mapping in logics can be combined with additional constructs such as validity, satisfiability, and conclusions. A expression of the construct ‘formula’ is considered to be true if it is mapped to the truth value \( true \) from \( LOG \). A expression of a syntactic type follows from a set of expressions if the truth value \( true \) is assigned to the expression whenever this truth value has been assigned to the set of expressions.

This understanding of logics in general may be defined on transparent intensional logics [DM00]. The association of syntactic types and semantic types is typically rather restricted. In some case, we might broaden this specific form of semantics.

Classical predicate logics is based on monotone reasoning. The derivation operator \( \vdash \) for a calculus \( \Gamma \) and the conclusion operators \( \models \) for the inheritance of validity in structures are monotone, i.e for sets of formulas \( \Sigma, \Sigma' \) and a formula \( \alpha \) we may conclude that \( \Sigma \cup \Sigma' \vdash \alpha \) or \( \Sigma \cup \Sigma' \models \alpha \) if \( \Sigma \vdash \alpha \) or \( \sigma \models \alpha \), correspondingly. Operators \( \mathcal{A}_\Phi \) define a closure for a set-formula relation \( \Psi \) on \( \mathcal{L} \), e.g., \( Cons(X) = \{ \alpha \in \mathcal{L} \mid X \models \alpha \} \) and \( Deriv(X) = \{ \alpha \in \mathcal{L} \mid Ax \cup X \vdash \alpha \} \) for a set of axioms \( Ax \).

The first main theorem of monotone reasoning states that \( \Sigma \cup Ax \vdash \alpha \) if and only if \( \Sigma \models \alpha \) for any set \( \Sigma \subseteq \mathcal{L} \) and for any formula \( \alpha \in \mathcal{L} \) (Completeness and soundness).

Useful properties for relationships \( \Psi \) and their closure operators \( \mathcal{A}_\Phi \) are

1. Reflexivity: \( \Sigma \subseteq \mathcal{A}_\Phi(\Sigma) \).
2. Monotonicity: If \( X \subseteq Y \) then \( \mathcal{A}_\Phi(X) \subseteq \mathcal{A}_\Phi(Y) \).
3. Closure: \( \mathcal{A}_\Phi(\mathcal{A}_\Phi(\Sigma)) = \mathcal{A}_\Phi(\Sigma) \).
4. Compactness: If \( \alpha \in \mathcal{A}_\Phi(X) \) then there exists a finite set \( X' \subseteq X \) such that \( \alpha \in \mathcal{A}_\Phi(X') \).
5. Inference property: If \( \alpha \rightarrow \beta \in \mathcal{A}_\Phi(X) \) then \( \beta \in \mathcal{A}_\Phi(X \cup \{ \alpha \}) \).
6. Deduction property: If \( \alpha \in \mathcal{A}_\Phi(X \cup \{ \beta \}) \) and \( \beta \) is closed then \( \beta \rightarrow \alpha \in \mathcal{A}_\Phi(X) \).
7. Generalization invariance: Let \( Gen(\Sigma) \) denote the set of generalization of formulas from \( \Sigma \), \( \mathcal{A}_\Phi(\mathcal{A}_\Phi(X)) = \mathcal{A}_\Phi(X) \).
8. Transitivity: If \( \alpha \in \mathcal{A}_\Phi(\Sigma) \) and \( \beta \in \mathcal{A}_\Phi(\Sigma \cup \{ \alpha \}) \) then \( \beta \in \mathcal{A}_\Phi(\Sigma) \).
The second main theorem of monotone reasoning states that if $A_{\varphi_1}$ and $A_{\varphi_2}$ are mappings from $2^L$ into $2^L$ with the property $A_{\varphi_1}(\emptyset) = A_{\varphi_2}(\emptyset)$ which fulfill the conditions 1. . . . 7. then for every $\Sigma \subseteq L$, $A_{\varphi_1}(\Sigma) = A_{\varphi_2}(\Sigma)$.

The third main theorem states that $Cons$ and $Deriv$ fulfill the conditions 1. . . . 7. and $Cons(\emptyset) = Deriv(\emptyset)$.

The fourth main theorem states that for any mapping $A_{\varphi} : 2^L \rightarrow 2^L$ the following assertions are equivalent:

(i). $A_{\varphi}$ is a compact closure operator (i.e. fulfills the conditions 1.,2.,3.,4.).

(ii). There exist a system $\vdash \Gamma$ of deduction rules on $L$ such that $A_{\varphi} \equiv A_{\vdash \Gamma}$.

The fourth theorem gives a very powerful characterisation of axiomatisability of closure operators. This characterisation covers axiomatisability of classical predicate logics and of Prolog-based predicate logics. The last one does not have the property of generalization invariance. It obeys the strong deduction property, i.e. if $\alpha \in A_{\varphi}(X \cup \{\beta\})$ then $\beta \rightarrow \alpha \in A_{\varphi}(X)$.

### 1.5 Formal Semantics in Computer Science

Computer science uses a variety of formal semantics:

**Operational semantics** interprets syntactic types by computational types of a possibly recursive machine may be based on varying context abstraction, and is often expressed in terms of state changes. Operational semantics is often defined on the basis of traces or sequences of states of the machine.

**Denotational semantics** associate mathematical functions to syntactic types, abstracts from context abstraction, and uses abstract states whose usage is based on the homomorphism principle. Denotational semantics uses partially ordered sets and their topology and thus defines Scott or Girard domains for interpretation.

**Axiomatic semantics** uses a calculus with axioms and proof rules that provides a mechanism for deriving correct syntactic types. Relational semantics is a special case of axiomatic semantics and is based on a specification of a start state and a set of final states. Functional semantics additionally assumes the set of final states to be singleton. Axiomatic semantics has also been defined through predicate transformers that provide a facility for backward reasoning starting with a target state.

**Transformational semantics** uses mappings to other syntactic types. They thus base semantics both on the (hopefully) well-defined mappings and on the semantics of the target system. Denotational semantics is a specific form of transformational semantics. It uses a mathematical formalism instead of another computer language. Categorical or functorial semantics is based on a translation to category theory.

**Algebraic semantics** uses a set of abstract basic syntactic systems with their semantics and a set of rules for construction of more complex systems based on these systems.

**Macro-expansion semantics** is based on static or dynamic inductive rewriting of syntactic types and allows to introduce abstractions such as the types of the $\lambda$ calculus. Abstraction may be directly defined or based on a notion of fixed points.
Grammar semantics is a specific form of operational semantics and uses a state consisting of semantic category variables and on instantiations for atomic non-terminal syntactic types.

These semantics can be modularised or layered. For instance, action semantics uses predefined expressions (action, data, and yielders) and denotational semantics.

Semantics is often mixed with “concrete semantics” [Bjø06] that provides and ‘everyday meaning’ and that is mixed with ‘pragmatic structures’. Often transformational semantics are used for interpretation of syntactic types. This interpretation is given in an informal way without providing a well-founded mapping and without precise semantics of the target domain. A typical example of such approach is the BPMN semantics that is based on informal description and on partial and changing mappings to BPEL types which are defined by various concrete implementations. Algebraic semantics are convenient if systems can be constructed from basic systems without limiting the construction process itself. Typical constructions of this kind are abstract data types, entity-relationship or object-relational database types, and frame types used in AI. It supports modularisation and separation of concern.

1.6 The Duality of Syntactic and Semantic Types

It is often claimed (e.g. [HR04]) that syntactic and semantic types cannot be defined. Mathematical logics uses a canonical approach to the definition of types. It starts with an enumeration of the alphabet. Words are constructed by certain rules over this alphabet. If elements of the alphabet can be categorised then we may consider similar alphabets and define a signature of elements of the language based on this categorisation. We may concentrate on all languages of the same signature. Computer science uses a similar approach by defining types through domains, a set of operations defined on the domains, and a set of predicates for the domains.

Formal languages have clearly defined and separated layers and use canonical techniques for construction of values of the language. We start with the enumeration of an alphabet. Some combinations of elements of these alphabets form words. Some of these words are considered to be reserved words which usage is restricted. The next layer groups words into expressions. The final layer constraints the expressions by additional conditions.

A syntactic type is given by one (or more) language(s), constructs, and a set of (concrete or abstract) expressions of interest. Let \( L \) be a set of (supported) languages, \( C_L \) the set of all constructs of a certain modelling language \( L \in L \) and \( S_C \) the set of all possible states of a certain construct \( C \in C_L \). An expression of the syntactic type \( T \) is a triple \((L, C, S)\) denoting a language \( L \in L \), a construct \( C \in C_L \), and an instance \( S \in S_C \).

Typically, one of the language is canonical. In this case we also consider a canonical set of constructs. We may omit the language and the set of constructs if we concentrate on the canonical language and set of constructs.

By a semantic type we understand a set of (concrete or abstract) semantic values within a certain (or some) language(s) and for a set of expressions: meanings of syntactic values. We may restrict the meaning of syntactic values to semantic values of the
same (or expanded) signature. We assume one special type \( LOG \) that provides truth values. We may also consider different variations or worlds. Let \( T_W \) a syntactic type of all worlds of interest. The set of possible worlds may also represent different versions of time and space.

The semantic type is typically defined by a state consisting of a set of individuals of certain categories, by operations and by predicates. The type \( LOG \) consists of truth values, connectives defined over these truth values, and generic quantifiers that provide an abstraction mechanism for some of its parameters to sets of values of other types. Examples of generic quantifiers are classical existence and generalisation quantifiers as well as generalised one- or two-dimensional quantifiers that are applied to pairs of expressions and describe the behaviour of components of the expressions in the whole. Typical generalised one-dimensional quantifiers are majority, many, at least and odd. Quantifiers may also be temporal ones.

In some cases we might also use signatures that are completely different. One of the misconceptions of formality in computer science is the belief that only such languages are formal that use mathematical symbols for expressions. Visual and diagrammatic languages might be formal as well.

The interpreter is based on an association of syntactic and semantic types and on an intensional function \( f : \omega_i_1 \times \omega_i_2 \times E \rightarrow \alpha \) which maps possible worlds \( (w_i_1, \ldots, w_i_n) \) and expressions \( E \) to elements of a semantic type \( \alpha \). Intensional functions may be evaluated but do not reveal the internal structure of the valuation or their relationship to other intensional functions. The intensional function can also be built by an intensional construction that is used to relate valuations of the function with valuations of other first order types. The context may either be a singleton context or a set of contexts. Given a context \( c \), a set of intensional functions \( \mathcal{F} = \{f_1, \ldots, f_n\} \), a logical language \( L \) and \( T \) a ‘theory’ of expressions from \( L \) that formulate the knowledge about the valuations of \( \mathcal{F} \). The tuple \( \mathcal{B} = (\mathcal{F}, T) \) is called a concept of a syntactic type in context \( c \). Let \( \mathcal{S} \) be a set of states. A configuration is a set of functions \( i : \mathcal{S} \rightarrow \mathcal{B} \) mapping states from \( \mathcal{S} \) to concepts from \( \mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\} \) in the current context.

The interpretation function can now be understood as a derived function for a syntactic type and a set of semantic types. Basic expressions are mapped according to the intensional function. Elaboration functions are assigned generic quantifiers.

### 1.7 Micro-Semantics of Wordings

Semantics often relies on pre-interpretation of some of its constructs and expressions. We may restrict the interpreter to a certain language, to a certain context and to certain states. In this case, the association of expressions to concepts becomes biased to a specific semantics. This micro-semantics is typically used with syntactic expressions such as words. The word has already a specific meaning. For instance, reserved words in programming languages are bound to a specific semantics.

This micro-semantics is often also used in applications of the so-called semantic web. Words are already provided together with a name space and a pre-interpretation. This pre-interpretation cannot be changed unless the name space is going to be changed. Communities agree on a common name space and thus introduce their ontology. The
sentence meaning of XML documents is exclusively based on the words it contains and their grammatical arrangement.

Micro-semantics of wordings is also used within the theory of word fields [ST08]. Words have their own syntactical ingredients, their micro-semantics, their language usage (or language game [Wit58]) and their pragmatic application area.

Micro-semantics should not be confused with semantics. It is only a part of semantics and cannot replace semantics or used to derive semantics. It is tempting to use such micro-semantics within a culture or community. It does however not provide the complete meaning of a syntactic expression.

2 Pearls of Database Theory

Designing a database application means to specify the structure, the operations, the static semantics and the dynamic semantics. The aim is to find a full specification or at least to find a specification which leads to a database structure on which operating is simple.

Structuring of databases is based on three interleaved and dependent semiotic parts [PBGG89, Tha91]:

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

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

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

2.1 Rigid Inductive Structures

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

A type constructor is a function from types to a new type. The constructor can be supplemented with a selector for retrieval (such as Select) and update functions (such as Insert, Delete, and Update) for value mapping from the new type to the component types or to the new type, with correctness criteria and rules for validation, with default rules, with one or more user representations, and with a physical representation or properties of the physical representation.
Typical constructors used for database definition are the set, tuple, list and multiset constructors. For instance, the set type is based on another type and uses algebra of operations such as union, intersection and complement. The retrieval function can be viewed in a straightforward manner as having a predicate parameter. The update functions such as Insert, Delete are defined as expressions of the set algebra. The user representation is using the braces {, }. The type constructors define type systems on basic data schemes, i.e. a collection of constructed data sets. In some database models, the type constructors are based on pointer semantics.

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

Then we define the structural recursion by insert presentation for \( R_C \) on \( T \) as follows

\[
\begin{align*}
\text{srec}_{h_0,h_1,h_2}(\emptyset_{CT}) & = h_0 \\
\text{srec}_{h_0,h_1,h_2}(\{s\}) & = h_1(s) \text{ for singleton collections } \{s\} \\
\text{srec}_{h_0,h_1,h_2}(\{s\} \cup_{CT} R_C) & = h_2(h_1(s), \text{srec}_{h_0,h_1,h_2}(R_C)) \text{ if } \{s\} \cap_{CT} R_C = \emptyset_{CT}.
\end{align*}
\]

All operations of the relational database model and of other declarative database models can be defined by structural recursion. Structural recursion is also limited in expressive power. Nondeterministic while tuple-generating programs (or object generating programs) cannot be expressed. We observe, however, that XML together with the co-standards does not have this property.

Another very useful modelling construct is naming. Each concept type and each concept class has a name. These names can be used for the definition of further types.

Static integrity constraints are specified within the universe of structures defined by the structural recursion.

Observation 1.
Hierarchical structuring of types leads to a generalized first-order predicate logics.

Observation 2.
In general, cyclic structuring leads to non-first-order logics. Structures with abstract linking are potentially cyclic.

2.2 Static Integrity Constraints

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

- **Component-construction constraints** are based on existence, cardinality and inclusion of components. These constraints must be considered in the translation and implication process.
- **Identification constraints** are implicitly used for the set constructor. Each object either does not belong to a set or belongs only once to the set. Sets are based on simple generic functions. The identification property may be, however, only representable through automorphism groups [BT99]. We shall later see that value-representability or weak-value representability lead to controllable structuring.
Acyclicity and finiteness of structuring supports axiomatisation and definition of the algebra. It must, however, be explicitly specified. Constraints such as cardinality constraints may be based on potential infinite cycles.

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

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

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

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

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

Often structures include also optional components. Let us denote the set of all components of a set $O$ of objects by $\text{compon}(O)$ and the set of all optional components of $O$ by $\text{compon}^{\text{opt}}(O)$. Similarly we denote the set of all components used in a constraint $\alpha$ by $\text{compon}(\alpha)$. Validity of constraints is either based on strong semantics requiring validity for all object sets independently on whether $\text{compon}^{\text{opt}}(O) \cap \text{compon}(O) \neq \emptyset$ or on weak semantics requiring validity for constraints only for those object sets $O$ for which $\text{compon}^{\text{opt}}(O) \cap \text{compon}(O) = \emptyset$. Classical validity is based on weak semantics which has a severe disadvantage:

Observation 3. Weak semantics leads to non-additivity of constraints for object sets $O$ with $\text{compon}^{\text{opt}}(O) \neq \emptyset$, i.e., it is not true in general that $O \models \{\alpha_1, \ldots, \alpha_m\}$ is valid if and only if $O \models \{\alpha_i\}$ for each constraint in $\{\alpha_1, \ldots, \alpha_m\}$.

Observation 4. Strong semantics leads to non-reflexiveness or non-transitivity of constraints for object sets $O$ with $\text{compon}^{\text{opt}}(O) \neq \emptyset$, i.e., $O \not\models \alpha \rightarrow \alpha$ for some constraints $\alpha$ or the validity of $O \models \alpha \rightarrow \beta$ and $O \models \beta \rightarrow \gamma$ does not imply $O \models \alpha \rightarrow \gamma$.

Since constraint sets may be arbitrary we might ask in which cases an axiomatisation exists. The derivation operator $\vdash_\Gamma$ of a deductive system $\Gamma$ and the implication operator $\models$ may be understood as closure operators $\Phi$, i.e.
(0) \( \Phi^0(\Sigma) = \Sigma \)

(i) \( \Phi^{i+1}(\Sigma) = \{ \alpha \in C \cap \Phi(\Phi^i(\Sigma)) \} \)

(+) \( \Phi^*(\Sigma) = \lim_{i \to \infty} \Phi^i(\Sigma) \)

for any subset \( \Sigma \) from a class \( C \) of constraints.

The closure operator \( \Phi \) is called compact for a class \( C \) if the property \( \alpha \in \Phi^*(\Sigma) \) implies the existence of a finite subset \( \Sigma' \) of \( \Sigma \) such that \( \alpha \in \Phi^*(\Sigma') \). It is called closed of \( \Phi^*(\Sigma) = \{ \alpha \in C \cap \Phi^i(\Phi^i(\Sigma)) \} \) for any \( \Sigma \subseteq C \). The closure operator is called monotone if \( \Phi^*(\Sigma) \subseteq \Phi^*(\Sigma \cup \Sigma') \). The operator is reflexive if \( \alpha \in \Phi^*(\Sigma \cup \{ \alpha \}) \) for all formulas and subsets from \( C \).

Observation 5.

The implication operator \( \Phi^* \upharpoonright \) is reflexive, monotone, closed and compact if and only if there exists a deductive system \( \Gamma \) such that \( \Phi^\Gamma \) and \( \Phi^* \upharpoonright \) are equivalent. If \( \Phi^* \upharpoonright \) additionally has the inference property, the deduction property and is generalization invariant then \( \Phi^\Gamma(\emptyset) = \Phi^* \upharpoonright (\emptyset) \).

If the deduction property fails then the axiomatisation by a deductive system may be based on some obscure rules similar to those for the axiomatisation of PROLOG.

Constructors used for construction of more complex types are often used for convenience and representing a different structuring. A typical example is the application of the list constructor with the meaning of representing sets. In this case we must add an list-to-set axiom

\[
\forall t \in \text{compon}(o)\forall i, j (\text{type}(o.i) = \text{type}(o.j) = t \Rightarrow \text{value}(o.i) = \text{value}(o.j))
\]

This axiom is often overseen and not considered.

Observation 6.

Semantics for structures defined by the list constructor and representing set must be extended by list-to-set axiom.

Since attributes are also constructed on the basis of constructors from base types we may ask whether this construction affects the definition of constraints and the axiomatisability. This question is open for most of the constraints. In [Lin03] it has, however, shown that keys and functional dependencies have a similar treatment as in the relational case. Substructures are, however, more complex and represented by the Brouwerian algebra of subcomponents.

2.3 Application of Database Semantics to Semantical Models

The entity-relationship model has been extended to the higher-order entity-relationship model (HERM)[Tha00a]. HERM is a set-theoretic based, declarative model which objects are value-oriented. For this reason, object identifiers are omitted.

The entity-relationship model uses basic (or atomic) data types such as \text{INT}, \text{STRING}, \text{DATE}, etc. the null type \( \bot \) (value not existing). Using type constructors for tuples, finite (multi-)sets and lists and union we construct more complex types based on standard set semantics:

\[
t = l : t | B | \{ a_1 : t_1, \ldots, a_n : t_n \} | \{ t' \} | ( t') | [ t']
\]

These types will be used to describe the domains of (nested) attributes.
Attributes allow to conceptually abstract from describing values. Associated data types provide the possible values. We use a set of names $N = N_0 \cup N_c$ for attributes such as address, street, city, name, first_name, destination, trip_course etc. Elements from $N_0$ are called atomic attributes.

Each atomic attribute is associated with a atomic type $\text{dom}(A)$.

Nested attributes are inductively constructed from simpler or atomic attributes by the iteration condition:

For already constructed nested attributes $X, X_1, \ldots, X_n$ and new attributes $Y, Y_1, \ldots, Y_m$, the sequences $Y(X_1, \ldots, X_n)$, $Y\{X\}$, $Y[X]$, $Y((Y_1 : X_1) \cup \cdots \cup (Y_n : X_n))$ are tuple-valued, set-valued, list-valued, multiset-valued and union-valued nested attributes.

Associated complex types are defined by the attribute structure. In the logical calculus below we use only tuple-valued and set-valued attributes. The calculus can similarly be extended.

For all types we use set semantics based on the basic type assignment.

Entity Types (or level-0-types) $E = (\text{attr}(E))$ are defined by a set $\text{attr}(E)$ of nested attributes. A subset $X$ of attributes can be used for identification. This subset is called key of $E$. In this case we consider only those classes which objects can be distinguished by their values on $X$.

Relationship Types (or level-$(i+1)$-types) $R = (\text{comp}(R), \text{attr}(R))$ are defined by a tuple $\text{comp}(R)$ of component types at levels $\leq i$ with at least one level-$i$-type component and a set $\text{attr}(R)$ of nested attributes. We use set semantics with expanded components under the restriction that $\text{comp}(R)$ forms a key of $R$. Unary relationship types with $|\text{comp}(R)| = 1$ are subtypes.

Clusters (also level-$i$-types) $C = C_1 \oplus \cdots \oplus C_k$ are defined by a list $(C_1, \ldots, C_k)$ of entity or relationship types or clusters (components). The maximal level of components defines level $i$. We set semantics (union) or equivalent pointer semantics.

Corresponding classes of a type $T$ are denoted by $T^C$. $\mathcal{R}(T)$ is the set of all classes of $T$. Basic type assignment is equivalent to pointer semantics with value representability.

The usual graphical representation of the extended ER model is a labelled directed acyclic graph. Entity types are denoted graphically by rectangles. Relationship types are graphically represented by diamonds with arrows to their components. Attributes are denoted by strings and attached to the types by lines. Key components are underlined.

A HERM scheme $S$ is a set $\{R_1, \ldots R_m\}$ of types of level 0, $\ldots, k$ which is closed, i.e. each set element has either no components (entity type) or only components from $\{R_1, \ldots R_m\}$.

Based on the construction principles of the extended ER model we can introduce the HERM algebra [Tha00a]. In general, for a HERM scheme $S$ the set $\text{Rec}(S)$ of all expressions definable by structural recursion can be defined.

Since HERM database schemes are acyclic and types are strictly hierarchical we can construct a many-sorted logical language by generalizing the concept of variables.

Given a HERM scheme $S$. Let $N_S$ the set of all names used in $S$ including type names. A sort is defined for each name from $N_S$. The sort sets are constructed according...
to the type construction in which the name has been used. Entity and relationship types are associated with predicate variables. The logical language uses type expansion for representation of relationship types. We can use key-based expansion or full expansion. Full expansion uses all components of the component type. Key-based expansion uses only (primary) key components. If all names are different in $S$ then we can use lowercase strings for variable names. If this is not the case then we use a dot notation similar to the record notation in programming languages.

The HERM predicate logic is inductively constructed in the same way as the predicate logic. Instead of simple variables we use structured variables. This language enables us to specify restrictions on the scheme. Queries can be expressed in a similar way.

We can also specify behavior of a database over lifetime. A database is modified by an action or more general by a transaction. Basic actions are queries or conditional manipulation operations. Manipulation operations such as insert, delete, update are defined in the HERM algebra. Database behavior can be specified on the basis of states. Given a HERM scheme $S = \{R_1, \ldots, R_m\}$. A state is the set of classes $\{R^C_{C_1}, \ldots, R^C_{C_m}\}$ with $R^C_i \in R(R_i), 1 \leq i \leq m$ which satisfies certain restrictions $\Sigma$.

The structuring of the extended ER model allows to deduct a number of properties. As an example we consider the axiomatisation of constraints generalizing those discussed in [Tha91]. We observe first that implication in the hierarchical predicate logic is reflexive, monotone, compact and closed. Let us consider classes of BV-constraints in HERM which form a cylindric algebra [Tsa89]. The order of constraints by $\Phi | = \text{possibly can be based on the order of of premises and conclusions. In this case the constraint set forms a pair algebra.}

Observation 7.
Cylindric classes are pair algebras.

Examples of cylindric classes are the class of functional dependencies, the classes of Hungarian functional dependencies [Tha91], the class of inclusion dependencies and the class of multivalued dependencies. Further, the $n$-class of all $\geq n$-functional dependencies $X \rightarrow Y$ which left side contains at least $n$ components and the class of rigid $\leq n$-inclusion dependencies $T_1[X] \subseteq T_2[X]$ which component list contain at most $n$ components form a cylindric constraint set. Usually, union does not preserve cylindric sets.

Observation 8.
Cylindric constraint classes are axiomatised by reflexivity axioms, augmentation and transition rules.

If an axiomatisation leads to reflexivity, augmentation and transitivity then union and decomposition rules can be deducted by the other rules. Transitivity may have to consider the specific influence of premises, e.g., transitivity for full multivalued dependencies is based on the root reduction rule [Tha91].

Based on this axiomatisation we may introduce a general vertical decomposition form:
Given a schema structuring $S = (ER, \Sigma_S)$. A vertical decomposition of $S$ is given a a mapping $\tau$ from $S$ to $S'$ which is defined by projection functions. The decomposition
is lossless if a query $q$ on $S'$ can be defined such that for each $db$ on $S$ the equality $q(\tau(db)) = db$ is valid.

Let further $\Sigma'$ the set of those constraints from $\Phi_{\text{In}}(\Sigma)$ which are entirely defined on the structures in $S'$. A decomposition based on projection is called $C$-constraint preserving if $\Sigma' \subseteq \Phi_{\text{In}}(\Sigma')$.

Classical example of vertical decompositions are decompositions of relations to relations in the third normal form.

We may now introduce a general class of $C$-decomposition algorithms:

- Construct basic elements which are undecomposable.
- Derive maximal elements by backward propagation of augmentation.
- Reduce redundancy in the constraint set by backward propagation of transitivity.
- Derive a left-right graph by associating conclusions of a constraint with the premise of another constraint.
- Combine all minimal left sides of constraints which are not bound by another constraint to a group.
- Derive projections based on all groups in the graph.

The first step of the decomposition algorithm is only introduced for convenience. This algorithm is a generalization of the classical synthesis algorithm.

Observation 9. The $C$-decomposition algorithm leads to $C$-constraint preserving decomposition if the class $C$ is cylindric.

### 2.4 Maturity and Capability of Object-Oriented/Object-Relational Models

Object-oriented database models have been developed in order to overcome the impedance mismatch between languages for specification of structural aspects and languages for the specification of behavioral aspects. So far, no standard approach is known to object-orientation. Objects are handled in databases systems and specified on the basis of database models. They can own an object identifier, are structurally characterized by values and references to other objects and can posses their own methods, i.e.

$$o = (i, \{v\}, \{ref\}, \{meth\})$$

The value characterization is bound to a structure of the type $T$ which is already defined. Characterizing properties of objects are described by attributes which form the structure of the object. Objects also have a specific semantics and a general semantics. The properties describe the behavior of objects. Objects which have the same structure, the same general semantics and the same operators are collected in classes. The structure, the operations and the semantics of a class are represented by types $T = (S, O, \Sigma)$. In this case, the modelling of objects includes the association of objects with classes $C$ and their corresponding value type $T$ and reference type $R$. Therefore, after classification the structure of objects is represented by

$$o = (i, \{(C, T, v)\}, \{(C, R, ref)\}, \{T, meth\})$$.
The recognized design methodologies vary in the scale of information modelled in the types. If objects in the classes can be distinguished by their values, then the identifiers can be omitted and we use value-oriented modelling. If this is not the case, we use an object-oriented approach. In the object-oriented approach, different approaches can be distinguished. If all objects are identifiable by their value types or by references to identifiable objects, then the database is called value-representable. In this case, the database can also be modelled by the value-oriented approach, and a mapping from the value-representable scheme to a value-oriented scheme can be generated. If the database is not value-representable, then we have to use object identifiers. In this case either the identifier handling should be made public or else the databases cannot be updated and maintained. Therefore, value-representable databases are of special interest. Thus, we can distinguish database models as displayed in Figure 1.

![Database Model Diagram]

**Fig. 1. Classification of Database Models**

It has been shown in [BT99,Sch94] that the concept of the object identifier can only be treated on the basis of higher-order epistemic and intuitionistic logics. Furthermore, identification by identifiers is different from identification by queries, equational logics and other identification methods. For this reason, the concept of the object identifier is far more complex than wanted and cannot be consistently and equivalently treated in database systems. Furthermore, methods can be generically derived from types only in the case if all objects are value-representable. Value-representable cyclic type systems require topos semantics [ST99] what is usually too complex to be handled in database systems. It can be shown that value-representable, non-cyclic type systems can be represented by value-oriented models.

### 2.5 XML - Couleur De Rose And Pitfalls

XML document specification and playout is based on a number of standards and co-standards:

**XML documents** are based on trees of elementary documents which are tagged by a begin and end delimiter.

**Document schema specification** is either based on

- **DTD specification** which supports an entity-relationship modelling,
RDF schema reuse which allows to include other specifications, or
Schema specification which allows object-oriented modelling.

XLink, XPointer, XPath and XNamespace support a flexible parsing and playout
enabling documents to be extended by foreign sources which might be accessible.

XSL supports filtering, union and join of documents.
XML query languages add query facilities to XML document suites.
XML applications are supported by a large variety of application standards such as
BizTalk and LOM.

This layering has a number of advantages and a number of disadvantages. The
main advantage of this approach is that almost any object set can be represented by
an XML document suite. XML documents may be well-formed. In this case they are
semi-structured and may be represented by $A$-trees which are defined by induction as follows

- each $\sigma \in A$ is a tree, and
- if $\sigma \in \Sigma$ and $t_1, \ldots, t_n$ are trees then $\sigma(t_1, \ldots, t_n)$ is a tree.

The set $\text{Dom}(t) \subseteq N^*$ of all nodes of a tree $t = \sigma(t_1, \ldots, t_n)$ is given by:
$\text{Dom}(t) = \{\epsilon\} \cup \{ui|i \in \{1, \ldots, n\}, u \in \text{Dom}(t_i)\}$
where $\epsilon$ is the root, $ui$ is the $i^{th}$ child of $u$, and
$\text{lab}(u)$ is the label of $u$ in $t$.

The disadvantages of XML stem from the generality of the approach. For instance,
parsing of XML document sets must be supported by machines which are not less com-
plex than Turing machines, i.e., tree automata
$M = (Q, \Sigma, \delta, (I_\sigma)_{\sigma \in \Sigma}, F), \quad F \subseteq Q, \delta : Q \times \Sigma \rightarrow 2^Q^*$.
A run $\lambda : \text{Dom}(t) \rightarrow Q$ specifies for each leave node $v : \epsilon \in \delta(\lambda(u1), lab^i(u))$ and for
each node $v$ with $p$ children : $\lambda(u1)\lambda(u2)\ldots\lambda(up) \in \delta(\lambda(u), lab^i(u))$.
The run accepts the tree $t$ if $\lambda(\epsilon) \in F$.

XML document suites have, however, a number of other properties: they are par-
tial and based on list semantics. Their implication is neither compact nor monotone nor
closed. Therefore, the axiomatisation of XML constraints is more difficult compared
with other database models. For instance, already the definition of keys and functional
dependencies becomes a nightmare. The treatment of cardinality constraints is more
difficult than for ER models. For instance, the definitions of [AFL02,BDF+01] are in-
complete since they do not consider the list-to-set axiom.

XML documents provide a universal structuring mechanism. [Kle07] has developed
a modelling approach that limits the pitfalls of XML specification.

Finite implication of path constraints is co-r.e. complete and implication is r.e. com-
plete for semi-structured models. The implication problem for key constraints is harder
than in the relational and ER case. It involves implication on regular path expressions
which is known to be PSPACE-hard. The satisfiability problem for key constraints and
referential inclusion constraints becomes undecidable in the presence of DTD’s. For
this reason, simpler language must be used for specification of constraints in XML.
3 Overuse and Misuse of the Notion of Semantics

Nowadays semantics is viewed in a large variety of ways. Some of them are [HR04]: semantic web, semantics as a metamodel, semantics as context, semantics as behaviour, semantics as being executable, semantics as the meaning of individual constructs, semantics as mathematical notation, and semantics as mappings. All these views have their own pitfalls and do not convey with the notion of semantics. They are mainly syntactic types that ‘cry out for’ [Bjø06] a semantic explanation.

3.1 Semantification of Semantic Web

The “Semantic Web” is mainly based on syntax and partially uses micro-semantics of wordings. Semantics is used in the sense of rudimentary lexical semantics. Rudimentary lexical semantics must be enhanced by explicit definitions of symbols or words used. These definitions can be combined with the name spaces that provide a source for the lexical units used in a web document. The semantification project [DHT+08] of the group headed by J. Pokorný and P. Vojas at Charles University Prague aims in enhancing the ontology based annotation in XML documents or RDFa-annotated HTML files by a semantic repository, by user profiles and by portfolio management.

Web documents should be enhanced by context [KSTZ03] or meta-data similar to the OpenCyc project. Lexical units may be characterised by time(absolut, type), place(absolut, type), culture, sophistication/security, topic/usage, granularity, modality/disposition/epistemology, argument preferences, justification, and lets [Len02].

The vocabulary of name spaces or of ontologies is not just a collection of words scattered at random throughout web documents. It is at least partially structured, and at various levels. There are various modes and ways of structuring, e.g., by branching, taxonomic or meronymic hierarchies or by linear bipole, monopolar or sequenced structures.

Ontologies are often considered to be the silver bullet for web integration. They are sometimes considered to be an explicit specification of conceptualisation or to be a shard understanding of some domain of interest that can be communicated across people and computers. We should however distinguish a variety of ontologies such as generic, semiotic, intention/extension, language, UoD, representational, context and abstraction ontologies.

3.2 Separation of Semantics and Behaviour

A common misconception in computer science is confusion of semantics and behaviour. Behaviour is an aspect in systems description. It is also based on syntax, semantics and pragmatics. We use dynamic algebraic structures of a certain signature for the description of behaviour.

Typical aspects concentrate either on models describing structural properties or evolution or the collaboration among actors or the distribution or architecture of the system. Figure 2 surveys aspects we might consider.

Aspects describe different separate concerns.
Structuring of a database application is concerned with representing the database structure and the corresponding static integrity constraints.

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

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

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

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

### 3.3 Semantics Illustration Through Diagrams

The Unified Modelling Language UML consists of more than 100 different types of diagrams. These diagrams provide some illustration for syntactic constructs. They are partially semantically founded. Their meaning is often only given by an abstract description of the intended meaning or by an ad-hoc polymorphism of constructs.

Diagrams make use of pictured elements. These elements are used in diagrams in very different ways and meanings. As an example one might compare the meaning of the arrow or the edge in different UML diagrams. The meaning of these elements is implicitly defined by the kind of diagram. The same problem can be observed for assignment of cardinality constraints to edges in ER diagrams despite the differences between participation and look-through semantics. Consistency of elements used in different diagrams is still an open issue.

Diagrams provide a simple way for visual representation of structures, functions, interaction or distribution. Visualization is, however, not the silver bullet as often marketed. It may mislead, e.g., by misleading comparisons or by overuse or wrong use of coloring schemes that vary in cultural environments. Representation of complex structures, e.g., in medicine cannot be entirely based on visual structures. Reasoning on representations, e.g., in UML diagrams is not yet supported. Diagrams may misguide as well as values without context.
The most important drawback of diagrams is their spatial restriction due to the requirement that diagrams must be surveyable. This requirement causes a separation of content into an ensemble of diagrams. The consistency of diagrams must thus be explicitly handled. So far there is no theory that allows to handle consistency in diagram ensembles such as UML diagram ensembles.

4 Contributions of SDKB to Data and Knowledge Base Semantics

This volume continues the ‘Semantics in Databases’ workshops. The first workshop has been published in Lecture Notes in Computer Science 1358 and the second in Lecture Notes in Computer Science 2582. The third workshop was collocated with EDBT 2008. We are thankful to the EDBT organisers in Nantes. At the workshop six papers selected from 19 papers that have been submitted. Two contributions that reflect the topic of semantics in databases and in knowledge bases have also been invited to this workshop. This volume additionally contains two invited papers.

Already with the second workshop we realised that the approaches to semantics in database research and in knowledge base research are similar and may enhance each other. The introduction [BKST03] surveys different approaches to semantics in database research and discusses a program to research on database semantics. Most of these research questions as well as the problems in [Tha87,Tha00a,LT98] are still open.

The papers in this volume reflect a variety of approaches to semantics in data and knowledge bases:

A. Altuna develops a model-theoretic semantics for the formalisation of context in knowledge bases. The approach accommodates reasoning across the different perspectives that may coexist on a particular theory or situation. This approach allows an explicit distinction between the context and the interpretation for formal semantics as discussed above.

D. W. Archer and L. M. L. Delcambre propose an explicit capture model for data integration decisions. This approach allows to handle data integration in a similar way as users do while learning the content of a document ensemble. Users evolve their understanding by restructuring, copying, pasting, abstracting and extending them by their context. The paper shows how this integration of lexical, logical, prototype and dynamic semantics supports entity resolution (record linkage, de-duplication, data identification) and de-resolution.

R. Caballero and Y. García-Ruiz, and F. Sáenz-Pérez develop a theoretical framework for debugging Datalog programs based on the ideas of declarative debugging. Whenever a user detects an unexpected answer for some query the debugger allows to track the cause for missing and wrong answers. The debugging mechanism is based on computation graphs and debugging circuits.

D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati and M. Ruzzi introduce a complete system for data integration that uses an ontology and description logic as the main vehicle. Data integration supports a uniform access to a set of heterogeneous data sources and does not require that the user knows implementation details. Lexical semantics can efficiently (LogSpace)
be combined in an with a conceptual architecture, comprising a global schema, the source schema, and the mapping between the source and the global schema.

F. A. Ferrarotti and J. M. Turull Torres investigate the connection between the concept of relational complexity and the restricted second-order logic $SO^{\omega}$. This logic allows to characterize the relational polynomial-time hierarchy. The existential fragment of $SO^{\omega}$ captures relational NP. Prenex fragments of $SO^{\omega}$ exactly correspond to the levels of the relational polynomial-time hierarchy.

S. Hartmann, H. Köhler, S. Link, T. Trinh, and J. Wang survey the variety of proposals for the notion of keys in XML documents. Typically XML documents are defined by application of list type constructor to basic types. Therefore, semantics is different from the case of typical database modelling languages. Additionally, XML documents have optional components. The notion of the key must thus be redefined. There are several notions for keys in XML documents. They are partially incomparable, differ in their properties and expressive power.

Stephen J. Hegner develops a general technique for establishing that the translation of a view update defined by constant complement is independent of the choice of complement. Views provide partial information. The treatment of semantics by views has been one of the open problems in [BKST03]. It is shown that the decomposition mapping for the pair of views is required not only to be bijective on states but also on the sentences which define the information content of the component views.

Gabriele Kern-Isberner, Matthias Thimm and Marc Finthammer develop a theory of truth values beyond classical two-valued semantics based on degrees of plausibility that express how strongly a formula is supported. Conditionals serve as a basic means to encode plausible, intensional relationships in the form of general rules. An algebraic theory of conditional structures is presented which provides the formal machinery to make use of conditionals for inference and belief revision in different frameworks. The approach is used for the development of a theory of qualitative knowledge discovery.

N. Mohajerin and N. Shiri use query rewriting techniques for the development of a top-down approach for answering queries using only answers to views. On the basis a graph-based model for conjunctive queries and views, the algorithm proposed in the paper efficiently generates maximally contained rewritings which are in general less expensive to evaluate, compared to the bottom-up algorithms, without requiring post-processing. Existing solutions that follow a bottom-up approach require a post-processing phase.

Héctor Pérez-Urbina, Boris Motik and Ian Horrocks develop a resolution-based algorithm for rewriting conjunctive queries over TBoxes in Description Logic DL-Lite+. The algorithm produces an optimal rewriting when the input ontology is expressed in the language DL-Lite. The complexity is NLogSpace w.r.t. data complexity. Combining this result with the lower bound it is concluded that query answering in DL-Lite+ is NLogSpace-complete.
References


Klaus-Dieter Schewe and Bernhard Thalheim

[Sch94] K.-D. Schewe. The specification of data-intensive application systems. Advanced PhD (Habilitation Thesis), Brandenburg University of Technology at Cottbus, Faculty of Mathematics, Natural Sciences and Computer Science, 1994.


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

Gunar Fiedler, Bernhard Thalheim
Department of Computer Science, Christian-Albrechts University at Kiel
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 Management

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.

**Content Management and Semantic Annotations**

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 $\langle S, P, O \rangle$ 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 modeler 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 \(\mathcal{C} = (D, I, T, O)\) 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 Structure Dimension

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, $\mathcal{S}_L$ the set of all schemata 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 $S \in \mathcal{S}_L$, and an instance $I \in \Sigma_S$.

In our example, $(\text{'HERM'}, s, i)$ 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 *List* with the entity set $\{\{\text{No} : 1\}\}$, the entity type *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 *consistsOf* with the relationship set $\{\{\text{List.No} : 1, \text{Product.No} : 134, \text{pos} : 1\}, \{\text{List.No} : 1, \text{Product.No} : 521, \text{pos} : 2\}\}$.

The structure dimension of content chunks is based on the theory of media types [?]. Media types [?] combine views and their functions into one type. Media types may be linked to other media types. For instance, we may distinguish input data for the workflow, retrieval data for the workflow, output data of the workflow, display data suites for each stage of the workflows, and escorting data supporting the understanding of each stage of the workflow.

Media objects may be structured, semi-structured, or unstructured by the media types. They are data that are generated from underlying databases, ordered, hierarchically representable, tailorable to various needs and enhanced by functionality for its usage. Since users have very different needs in data depending on their work history, their portfolio, their profile and their environment media types are packed into containers. Containers provide the full functionality necessary for the application and use a special delivery and extraction facility. The media type suite is managed by a system consisting of three components:

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

**Media object storage and retrieval system:** Media objects can be generated on the fly whenever we need the content or can be stored in the storage and retrieval subsystem. Since their generation is usually complex and a variety of versions must be kept, we store these media objects in the subsystem.
**Media object delivery system:** Media objects are used in a large variety of tasks, by a large variety of users in various social and organizational contexts and further in various environments. We use a media object delivery system for delivering data to the user in form the user has requested. **Containers** contain and manage the set of media object that are delivered to one user. The user receives the user-adapted container and may use this container as the desktop database.

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

**The Topic Dimension**

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, ?]) 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.

Figure 3: Topic Map for the ‘Product List’ Content Chunk

Topic parts of a content chunk thus serve for several purposes: to represent ideas or understandings; to represent a complex structure of content chunks and their related chunks; to communicate complexes of content chunks; to aid understanding by explicitly integrating new and old content chunks; and to assess understanding or diagnose misunderstanding.

The Origin Dimension

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 $\mathcal{U}$ 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 $\mathcal{U}$ be a universe of contexts and let $C \subseteq \mathcal{U}$ be a finite set of contexts. Further, let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a set of content chunks. The origin part of a content chunk $\mathcal{C}$ is a tuple $(c, A)$ with a context $c \in C$ where the content chunk $\mathcal{C}$ 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 origin dimension also reflects the purpose of the content chunk. Content chunks have a function such as to give an instruction, to control behaviour, to transmit ideas in a memorable form, to enhance social cohesion, or to give users a better understanding. For example, a content chunk that reflect a piece of knowledge may start with a mystery that leads to a conflict situation, may continue with an explanation of the solution or of the discovery as the turning point and may conclude with a resolution of the conflict situation. Context injection must thus be an integral element for content chunk processing.

Our origin model for content chunks extends the usage thesis of that mainly reflect the communication act between a sender and receiver with their intentions, backgrounds, cultures and relationships. Usage context should also consider excluded receivers, value of content chunks to receivers, groups or societies. Content chunks are thus generically enhanced by context, refined by intended specifics and instantiated by their specific usage.

The Intension Dimension

The intention dimension is based on concepts. They are the building blocks in human thinking and reasoning, and as such highly flexible. They can be general or specific, concrete or abstract, natural or technological, artistic or scientific. They can apply to things that are real or imaginary. They provide a help for distinguishing between things we observe in the world, or ideas such as truth and falsity, appearance and reality and continuity and discontinuity. Abstract concepts are useful for characterisation of observations, thoughts and expressions. Typical abstract concepts are truth and falsity, sameness and difference, wholes and parts, subjectivity and objectivity, appearance and reality, continuity and discontinuity, sense and reference, meaningful and meaningless and problem and solution. They govern different kinds of human thinking at a fundamental level.

Concepts are vital to the efficient functioning of semantic Wikis. They are organised bundles of stored knowledge which represent an articulation of events, entities, situations, and so on experience. Concepts are necessary for an understanding, for the organisation, for sharing Wikis and for communication. We may assume a simple association between the components of Wikis and concept. The associations may form a complex multi-dimensional network. They may be of specific types such as kind-of, is-part-of, is-used-for and of variable strength. Associations typically correspond to concepts of a more schematic kind than the concepts which they serve to connect.

The classical approach to concepts is based on description of necessary and sufficient criteria for content-concept association. We notice however that most concepts characterising content chunks cannot be captured by means of a set of necessary and sufficient features. Many natural concepts are fuzzy and contextually flexible. Therefore we need to extend the approaches typically assumed for formal semantics to natural semantics. Additionally, the association of content to concepts must not be strict. Some content may be a better example to a concept than other content.

The prototype approach for concept-content association is also limited. Ratings or se-
lections of prototypes are strongly context dependent, e.g., culture dependent and actor dependent. Prototypes are given with certain preference, frequency, sample extraction, learning background, level of verifiability, and under time pressure. The degree of association may vary over time, may be dependent on the concrete usage, and bound by the representation language chosen. Prototype content may also be more or less specific or general for concepts.

Concepts are typically expressed through propositions. The meaning has typically two parts: an element of assertion and something that is asserted. What is asserted is called proposition. The simplest type of proposition consists of an argument and a predicate. Semantical units or propositions are interrelated by entailment. Entailment is different from material implication and relates propositions by forward propagation of truth and backward propagation of falsity. Propositions can be contraries, contradictories, or independent. They may belong to a category or genre of expression, are given in a certain style or manner, are often based on stereotypical norms of expression, depend on ideas and values that are employed to justify, support or guide the expression, reflect aspects of culture or social order, are shaped according to the community that uses them, and are configured by theories or paradigms.

We also may distinguish between the existential approach to meaning based on a correlation of expressions in a language with aspects in the world. The intentional approach associates some kind of representation with concepts as the main constituents of the sense and depends on the cultural context. Whenever content is difficult to interpret then we need to consider concepts, deep structures, unconscious foundations, hidden symbols, annotations or underlying pattern supporting it. If content seems to transparent then we do not need to look for these things. It is often surprising how much background information is necessary for understanding content even such content that appear on the surface to be wholly transparent. There are various connotations and denotations that content may have. We have to consider the arrangements and laws for constructing content phenomena (langue) as well as the various instances that are constructed by constructors and laws (parole). Content can be coded in various ways, e.g. based on different representation such as text or multimedia elements. Content can be differently categorized and organised. We may use conventions that draw on common forms of knowledge. Furthermore, we need to devise different ways for understanding and for association of concepts to content.

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 $\iota_1, \ldots, \iota_n$ 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 ($\text{true}, \text{false}$) 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:
  - $\iota, o, \omega_i$ 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 $f : \omega_i \times \omega_k \rightarrow \alpha$ mapping possible worlds $(w_1, \ldots, w_k)$ to instances of a first order type $\alpha$. An intensional function is called non-trivial if there are two possible worlds $(w_1, \ldots, w_k), (v_1, \ldots, v_k)$ with $f(w_1, \ldots, w_k) \neq f(v_1, \ldots, v_k)$.

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): $\text{isCustomer} : \omega \times \tau \rightarrow o$.

The valuation of this function may be determined by coupling the function with the customer database of our Web shop: $\text{isCustomer}(w, t) = \text{true}$ 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 $c \in C$. 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 $p : \mathcal{X} \to \mathcal{F}$ from variables $\mathcal{X} = \{x_1, \ldots, x_n\}$ to intensional functions $\mathcal{F} = \{f_1, \ldots, f_m\}$ 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 monotone 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, $\mathcal{F} = \{f_1, \ldots, f_n\}$ a set of intensional functions, $L$ a logical language and $T$ a theory with sentences from $L$ formulating the knowledge about the valuations of $\mathcal{F}$ with respect to the layering of intensional functions in $c$. The tuple $\mathcal{B} = (\mathcal{F}, T)$ is called a concept in context $c$.

In our Web shop example we might consider intensional functions $isCustomer : \omega \times \tau \to o$, 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:

$$isCustomer(w, t) \implies (\forall t' > t)(isCustomer(w, t'))$$

In another example shopping carts ($isShoppingCart : \omega \times \tau \to o$) might become an order list ($isOrderList : \omega \times \tau \times \eta \to o$ for possibilities $\eta$):

$$isShoppingCart(w, t) \implies (\exists n' \in \eta, t' \in \tau)(isOrderList(w, t', n'))$$

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 $i : \mathcal{S} \to \mathcal{B}$ mapping facets from $\mathcal{S}$ to concepts from $\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ 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 $D$ for the set of all structure definitions in the state of the CMS, $I$ for the set of all defined intensions, $T$ the set of all defined topic maps, and $C$ 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 $\text{explain} : D \rightarrow 2^I$ returns all intensions associated with a certain data instance.
- $\text{sample} : I \rightarrow 2^D$ returns all data instances associated with a certain intension.
- $\text{express} : D \times I \times C \rightarrow 2^T$ returns all topic maps associated with the given data object under the given intension in the given context.
- $\text{understand} : T \times C \rightarrow 2^{I \times D}$ returns data instances together with an intension for the given topic map and context.
- $\text{find} : C \rightarrow 2^C$ 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.

![Figure 4: Imported Concepts, Constructions, and Local Theories](image-url)

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 Schemata

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 \( \text{generate} : T \times C \rightarrow 2^{D \times I} \), a function \( \text{deliver} : I \times D \times C \rightarrow T \), and a function \( \text{exchange} : D \times I \times T \times C \times C \rightarrow D \times I \times T \times C \).

These functions are defined for each context separately. First, a set of base intensions is defined. These base intensions rely on concepts (such as \textit{customer} or \textit{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 \( \{k_1, ..., k_n\} \) be a set of intensional constructions. An intensional construction template is a tuple \( (\{k_1, ..., k_n\}, p, m) \) with intensional constructions \( k_i \) for each facet in the intension specification frame, a parameter assignment specification \( p : X \rightarrow B \) mapping variables from \( B = \{x_1, ..., x_k\} \) to concepts from \( B = \{B_1, ..., B_l\} \) restricting valuations for variable substitutions in \( \{k_i\} \) to the given concepts, and a merging function \( m \) which creates

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

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.

A wiki is a collaborative web site set up to allow user editing and adding of content by any user who has access to it. Wikis have changed access and habits of internet users. The right and wrong usage of wikipedia is already widely studied in literature, e.g. the journal \textit{First Monday} provides a detailed study of Web 2.0 in issue 13, March 2008 and of wiki misuse in more than a three-score papers in various issues. The main functionality provided for wikis is

- management of content chunks with their structure, intention, origin and topic,
- annotation management for users depending on their role, rights and obligations,
- explanation, exploration and knowledge elicitation and gathering support for readers of content chunks,
- presentation of information in a variety of ways and for a variety of environments,
- user management depending on the roles, functions and rights a user may have on the content,
• security and safety management for integrity of content and information, and
• history and evolution management that allows to show the development of the wiki and to restore previous versions.

We concentrate the remaining part of the paper to the first three main functionalities of wiki systems. These three functionalities are backed by our approach to semantic wikis. Presentation may also include generic adaptation to the user environment and features for marking content [?]. Wiki systems should integrate features that have been developed for customer management. Wikis are generally designed with a functionality that makes it easy to correct mistakes. Since this functionality is a target for attacks on content and on the system, wiki systems are extended by security and quality management features. Thus, they provide a means to verify the validity of recent additions, changes, corrections, replacements etc. to the content. History and development information can be maintained through docket[s] [?] and the diff feature that highlights changes between two revisions. Wiki systems are special web information systems. They support information seeking life cases [?], storyboards for creation and consumption of information [?] and require a sophisticated user profile and portfolio model [?].

Wiki systems share and encourage several features with generalized content management systems, which are used by enterprises and communities-of-practice. They are maintained, developed and enriched by communities of leisure, interest or practice. Community members are allowed to instantly change the content (usually Web pages.) There are some minimal requirements to the content chunk for wikis. The name or annotation of a content chunk is typically embedded in the hyperlink and interlinked with other chunks. Content chunks can be partially created or edited at anytime by anyone (with certain limitations for protected articles). They are editable through the web browser. Their evolution history is accessible through a history/versioning vie, which also supports version differencing (“diff”), retrieving prior versions and summary of most recent additions/modifications. Additionally, easy revert of changes is possible. 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.

Semantic wikis\(^1\) enhance content that is displayed in the web with fully considered and perfected concepts or verified knowledge and with user annotation or topics. It thus formalises the notion of wikis enhanced by ontologies [?], clarifies the knowledge basis and provides a basis for using data from the web in a form that corresponds to the user demands, their information portfolio and their personal profile.

Using Communities for Content Generation

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. One of the most severe requirement to wiki evolution is trustworthiness of the wiki. Everybody

\(^{1}\) Our concept of semantic wikis should not be mistaken as a concept of Web 3.0. Web 3.0 or semantic web aims in annotation of content on the basis of an ontology which is commonly accepted by a community. Proposals such as the Sematic MediaWiki add a fixed annotation to concepts similar to tagging in websites such as delicio.us.
who uses wiki systems such as wikipedia observes that the competence level, the education profile, the work portfolio, the biases, the intensions (e.g., trolling) and the psychographical level of wiki creators has a huge impact on the quality of a wiki. Wikis that can be created by almost everybody are typically of lower quality than those that can only be created by experts in the area. As an example we accessed the entry ‘Dresden’ in wipipedia.org. In the average, each second sentence was not completely correct.

Wiki systems 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.

Modelling Wiki Functionality Based on Content Chunks

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 \( \mathcal{A} \) 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.

![Topic Map with Task Specification](image)

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:

\[
isProofReadComment(w) := (\exists w', t < now)(\text{moderator}(w, t) \land \text{proofread}(w, w'))
\]
From Wikis to Semantic Wikis: Extended Functionality

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


Development of Collaboration Frameworks for Web Information Systems

Klaus-Dieter Schewe 1, and Bernhard Thalheim 2
1 Massey University, Information Science Research Centre, Private Bag 11222, Palmerston North, NZ
2 Christian Albrechts University Kiel, Institute of Computer Science, Olshausenstr. 40, D-24098 Kiel
k.d.schewe@massey.ac.nz thalheim@is.informatik.uni-kiel.de

Abstract

Web information systems aim in support of collaborative work beyond that what is provided through distributed systems. Classical techniques for conceptual description of distribution concentrate on structures and distribution of data. Collaboration is not only based on structures but follows computations and business needs. Thus, we need to consider structuring, functionality and distribution at the same time. These aspects are intertwined with each other. Systems cooperate, communicate and coordinate their behaviour. We base our consideration on collaboration that integrates communication, coordination and cooperation. We develop a novel specification framework for collaborating systems in this paper.

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.

Collaboration adds to modelling two new dimensions: ubiquity and location. 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. 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

Although a number of approaches to collaboration has been proposed and experienced in the last two decades, modern 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. Typical additional semantical constraints are accept-on conditions [Srinivasa, 2000] or more generally rely-conditions and guarantee-conditions to both pre- and post-conditions. Rely-conditions specify the content and functionality on which a collaborating party may rely. Guarantee conditions specify which content and functionality is guaranteed by the service-providing party. We envision in this paper that these conditions can be generalized to a specific style of assumption-commitment specification.

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 [Thalheim, 2002]. We also consider small and evolving components.

Currently **massively collaborating systems** are developed and compete with classical systems. They are superseding 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 (3K or in English 3C stands for communication, cooperation, coordination) model for specification of distribution and collaboration. Collaboration is going to supported on the basis of exchange frames and information service [Lockemann, 2003]. The first specify dissemination, e.g., announcement, volume, time, and protocols. The latter are use for specification of the information service with extraction, transformation, load, and representation. Such distributed services are based on classical communication facilities such as routing, e.g., P2P like with query based network propagation), such as subnetting and propagation.

In this paper we enhance the information modelling language HERM [Thalheim, 2000] and the (web) storyboarding language SiteLang [Düsterhöft and Thalheim, 2001]. We introduce a novel language that allows to specify collaboration based on the 3K model. This languages neatly integrates with SiteLang and HERM.

### 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 Modelling of Collaboration

Figure 1 displays the three perspectives of collaboration. **Communication** is defined via exchange of messages and information or simply 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 rules collaboration. The specification is based on the pre-/post-articulation of tasks and on the description management of tasks, objects, and time. Coordination may be based on loosely or tightly integrated activities, may be enabled, forced, or blocked.

**Cooperation** is the production of work products taking place on a shared space. It can be considered as the workflow or life case perspective. We may use a specification based on storyboard-based interaction [Srinivasa, 2000] that mapped to (generic and structured) workflows. The information exchange is based on media types for production, manipulation, organization of contributions.

This understanding has become now a common model for collaboration. We use these ingredients of the perspectives for the specification of collaboration.

![Figure 1: The collaboration triangle relating communication, coordination, and cooperation](image)

A number of models have already been proposed for CSCW systems such as coordination theory [Malone and Crowston, 1994], activity theory [Kaptelinin et al., 1995], task management approaches [Kreifelts et al., 1999], action/interaction theory [Fitzpatrick et al., 1995], and object-oriented conceptual models [Teege, 1996]. We generalize these approaches and propose a more general model. We find dependent views on the diagram in Figure 1:

- **Communication act view** which is based on sending and receiving collaboration acts;
- **Concurrency view** which is based on commonly used data, functions, and tools;
- **Cooperation context view** that combines the context of cooperation, i.e., portfolio to be fulfilled, the cooperation story and the resources that are used.

#### 2.2 The Collaboration Style and Pattern

The **collaboration** style is based on four components describing supporting programs 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;
- **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, processor, asynchronous completion token, accept connector), synchronization (scoped locking, strategized locking, thread-safe interface, double-checked locking optimization) and parallel execution (active object, monitor object, half-sync/half-async, leader/followers, thread-specific storage):

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

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

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

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

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

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

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: | (characterization of initial states) |
| Initial state: | (characterization of target states) |
| Target state: | (profile presupposed for solution) |
| Profile: | (list of instruments for solution) |
| Instruments: | (list of auxiliary conditions) |
| Collaboration: | (collaboration style/pattern) |
| Auxiliary: | (final state, target conditions) |
| Execution: | (list of activities, control, data) |
| Role: | (general description) |
| Part: | (behavioral categories/stereotypes) |
| Communication: | (general description) |
| Coordination: | (protocols, services and exchange) |
| Cooperation: | (contracts and enforcement) |
| Restrictions: | (general description) |
| Environment: | (general description) |

2.4 Cooperation Specification Based On 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 execution, and iteration. We may derive extended operations such as simple iteration and optional execution. 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:

\[
\begin{align*}
T &; ((D \Box (C ; P)) \parallel (I ; U)) ; H : (R ; H ;) * ; \\
A &; (S \Box skip) ; E ; I ; N ; H ; (R ; H ;) ; A) * ; \ S \end{align*}
\]

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),
Coordination supports the consistency of work products, of work progress, and is supported by an explicitly specified coordinator. If work history is of interest, a version manager is integrated into the exchange support system. The coordination is supported by an infrastructure component. The coordination component observes modification of data that are of common interest to collaborating parties and resolves potential conflicts. The conflict resolution strategy is based on a cooperation contract. The contract is global to all parties and may contain extensions for peer-to-peer collaboration of some of the parties.

Coordination is based on a coordination contract. The contract consists of

- the coordination party characterization, their roles, rights and relations,
- the organization frames of coordination specifying the time and schema, the synchronization frame, the coordination workflow frame, and the task distribution frame,
- the context of coordination, and
- the quality requirements (ubiquity, security, interpretability, consistency, view consistency, scalability, durability, robustness, performance) for coordination,

Contract: (name)
Based on: general conditions
Parties: (general description)
Proprietor: (...) Possessor: (...) Trustee: (...) Arbiter: (...) Subject matter: (Media object suite)
Exchange: (binding obligations, permissions) Computation: (obligations, permissions)
Distribution: (obligations, permissions) Monitoring: (managers: recognizer, ...)
Notification: (states, timer, constraint scanner) Correlation: (protocols, obligations, permissions)
Considerations: (legal conditions) Enforcement: (actions, termination)

We distinguish four levels of coordination specification. The syntactical level uses an IDL description and may use coordination constructs of programming languages. We use constructs of the JDL (job description language) for this description of resources, obligations, permissions, and restrictions. The behavior level specifies failure-atomicity-, execution-atomicity- pre-, rely-, guarantee- and post-conditions. The synchronization level specifies service object synchronization and paths and maintains a synchronization counter or monitor. The fourth level specifies quality of services level.

The coordination profile is specified by a coordination contract, a coordination workspace, synchronization profile, coordination workflow, and task distribution.

Coordination profile: (name)
Based on: general conditions
Formation: (general description)
Contract: ...
Lifespan: ...
Contract variant: ...
Parties: (names)
Organization: (names, general description)
Infrastructure: (names, 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:

Party: (names)
Characteristics: ...
Profile: ...
Roles: ...
Rights: ...
Relations: ...
Part: (general description)
Organization: (general description)
Infrastructure: (general description)

Parties are usually organized within organizations such as groups:
Organization: (name)
Synchronization: ...
Stories: ...
Hierarchy: ...
Time slot: ...
Task distribution: ...
Coordination: name
Infrastructures: (names)

Party profiles simply use the frame:

Party profile: (party profile name)
Information demand: (general description)
Utilization pattern: (general description)
Specific utilization: (general description)
Party context: (general description)
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 $(S, Exchange Frames)$.

Communication contracts specify the collaboration architecture and the style of exchange.

3.1 Services

A service consists of a media type, the characteristics provided and properties guaranteeing service quality and is defined by the quadruple $(Media type, Service Manager, Competence, Characteristics)$, i.e. $S = (M, Man, C, F)$.

Media types offer their own functions including statistical packages, functions proposed for data warehouses, or data mining algorithms. The services manager $Man$ supports functionality and quality of services and manages containers, their play-out and their delivery to the client. It is referred to as a service provider. The competence of a service manifests itself in the set of tasks $T$ that may be performed and the guarantees for their quality.

Service: <!--name-->
Based on: <!--general conditions-->
Media types: <!--general description-->
  Raw media type: ...
  Extensions: ...
  Unit: ...
  Order: ...
  Co-/Adhesion: ...
  Hierarchy: ...
  Playout: ...
Services manager: <!--general description-->
  Kind: ...
  Communication: ...
  Coordination: ...
  Cooperation: ...
Competence: <!--general description-->
  Task: ...
  QoS: ...
The context of a service is characterized as follows:

Context: <!--name-->
  Media types: ...
  Environment: ...
  Range of variation: ...

3.2 Exchange Frames

Exchange frames might by 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. The left and middle part have already been discussed in the literature. The layering and the right part is our contribution. The later allows to develop implementations and realisations.

![Figure 2: Layers of a typical collaboration system](image-url)

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.

The data structures in Figure 3 allow a consistent treatment of collaborating systems based on the architecture in Figure.
2. In our website projects we experimented with a number of approaches supporting session management of collaborating societies.

This structure has successfully been used in the SeSAM system that supports the work of parliamentarians and parliaments. The collaboration in parliaments is rather complex. It consists in well-specified cooperation and coordination rules from one side and in rather chaotic volatile multi-party collaboration. Both kinds require sophisticated session and workspace support.

4 Conclusion

The research reported in this paper aims in the development of a general specification framework for collaborating web information systems. The proposed framework generalises approaches known so far by providing an integrated specification. At the same time the framework is based on a sound theory. 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


The list of references includes only those that are needed for understanding our framework. We decided to provide a brief list due to space limitations. A long list of references can be found on our websites.
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

Entity types are represented graphically by rectangles. Attribute types are associated with the corresponding entity or relationship type. Attributes primarily identifying a type are underlined. Relationship types are represented graphically by diamonds and associated by directed arcs to their components. A cluster type is represented by a diamond, is labelled by the disjoint union sign, and has directed arcs from the diamond to its component types. Alternatively, the disjoint union representation ⨁ is attached to the relationship type that uses the cluster type. In this case directed arcs associate the ⨁ sign with component types. An arc may be annotated with a label.

The definition scheme for structures.

The extended entity-relationship model uses a data type system for its attribute types. It allows the construction of entity types \( E \Vdash (\text{attr}(E), \Sigma_E) \) 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 \( T \Vdash \text{def} \).

The EER model lets users inductively build relationship types \( R \Vdash (T_1, \ldots, T_n, \text{attr}(R), \Sigma_R) \) 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 \( C \Vdash T_1 \cup \ldots \cup T_n \) 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:

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

**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 ($\langle \ldots \rangle$), set constructor ($\{ \ldots \}$), and the list constructor ($\ldots$). Typical base types are integers, real numbers, strings, and time. Given a set of names $N$ and a set of base types $B$, a basic attribute type $A :: B$ is given by an (attribute) name $A \in N$ and a base type $B$. The association between the attribute name and the underlying type is denoted by ::. The base type $B$ is often called the domain of $A$, i.e. $dom(A) = B$. Complex attributes are constructed on base attributes by application of the type constructors. The notion of a domain is extended to complex attributes, i.e. the domain of the complex attribute $A$ is given by $dom(A)$. Components of complex attributes may be optional, e.g., the $Title$ in the attribute Name.

Typical examples of complex and basic attributes in Figure 1 are

$\begin{align*}
Name & \triangleq (\text{FirstNames} < \text{FirstName}>, \text{FamName}, [\text{AcadTitles}], [\text{FamilyTitle}]) , \\
PersNo & \triangleq \text{EmplNo} \cup \text{SocSecNo} , \\
\text{AcadTitles} & \triangleq \{\text{AcadTitle}\} , \\
\text{Contact} & \triangleq (\text{Phone}([\text{PhoneAtWork}], \text{private}), \text{Email}, \text{URL}, \text{WebContact}, [\text{Fax}([\text{PhoneAtWork}])] ) , \\
\text{PostalAddress} & \triangleq (\text{Zip}, \text{City}, \text{Street}, \text{HouseNumber}) \\
\text{for DateOfBirth} & :: \text{date}, \text{AcadTitle} :: \text{acadTitleType}, \text{FamilyTitle} :: \text{familyTitleAcronym}, \text{Zip} :: \text{string7}, \\
\text{SocSecNo} & :: \text{string9}, \text{EmplNo} :: \text{int}, \text{City} :: \text{varString}, \text{Street} :: \text{varString}, \text{HouseNumber} :: \text{smallInt}.
\end{align*}$

The complex attribute Name is structured into a sequence of first names, a family name, an optional complex set-valued attribute for academic titles, and an optional basic attribute for family titles. Academic titles and family titles can be distinguished from each other.

**Entity types and entity classes.**

Entity types are characterized by their attributes and their integrity constraints. Entity types have a subset $K$ of the set of attributes which serve to identify the objects of the class of the type. This concept is similar to the concept of key known for relational databases. The key is denoted by $ID(K)$. The set of integrity constraints $\Sigma_E$ consists of the keys and other integrity constraints. Identifying attributes may be underlined instead of having explicit specification.

Formally, an entity type is given by a name $E$, a set of attributes $attr(E)$, a subset $id(E)$ of $attr(E)$, and a set $\Sigma_E$ of integrity constraints, i.e.

$$ E \triangleq (attr(E), \Sigma_E). $$

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

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

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

Objects on type $E$ are tuples with the components specified by a type. For instance, the object (or entity) (HRS3, 408A, 15) represents data for the Room entity type in Figure 1.
An entity class $E^C$ of type $E$ consists of a finite set of objects on type $E$ for which the set $\Sigma_E$ of integrity constraints is valid.

Cluster types and cluster classes.
A disjoint union $\cup$ of types whose identification type is domain compatible is called a cluster. Types are domain compatible if they are subtypes of a common more general type. The union operation is restricted to disjoint unions since identification must be preserved. Otherwise, objects in a cluster class cannot be related to the component classes of the cluster type. Cluster types can be considered as a generalisation of their component types.

A cluster type (or “category”)

$$C \doteq l_1 : R_1 \cup l_2 : R_2 \cup ... \cup l_k : R_k$$

is the (labelled) disjoint union of types $R_1, ..., R_k$. 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 $R_1^C, ..., R_k^C$. It is defined if $R_1^C, ..., R_k^C$ are disjoint on their identification components. If the sets $R_1^C, ..., R_k^C$ are not disjoint then labels are used for differentiating the objects of clusters. In this case, an object uses a pair representation $(l_i, o_i)$ for objects $o_i$ from $R_i^C$.

Relationship types and relationship classes.
First order relationship types are defined as associations between entity types or clusters of entity types. Relationship types can also be defined on the basis of relationship types that are already defined. This construction must be inductive and cannot be cyclic. Therefore, an order is introduced for relationship types. Types can only be defined on the basis of types which have a lower order. For instance, the type $Professor$ in Figure 1 is of order 1. The type $ProposedCourse$ is of order 2 since all its component types are either entity types or types of order 1. A relationship type of order $i$ is defined as an association of relationship types of order less than $i$ or of entity types. It is additionally required that at least one of the component types is of order $i - 1$ if $i > 1$. Relationship types can also be characterized by attributes. Relationship types with one component type express a subtype or an Is-A relationship type. For instance, the type $Professor$ is a subtype of the type $Person$.

Component types of a relationship type may be labelled. Label names typically provide an understanding of the role of a component type in the relationship type. Labelling uses the definition scheme $Label : Type$. For instance, the $Kind$ entity type is labelled by $Proposal$ for the relationship type $ProposedCourse$ in in Figure 1. Cluster types have the maximal order of their component types. Relationship types also may have cluster type components. The order of cluster type components of a relationship type of order $i$ must be less than $i$.

Component types that are not used for identification within the relationship type can be optional. For instance, the $Room$ component in Figure 1 is optional for the type $PlannedCourse$. If the relationship object in the $PlannedCourse$ class does not have a room then the proposal for rooms in $ProposedCourse$ is accepted. A specific extension for translation of optional components may be used. For instance, $Room$ in Figure 1 is inherited to $PlannedCourse$ from $ProposedCourse$ if the $Room$ component for a $PlannedCourse$ is missing.

Higher order types allow a convenient description of types that are based on other types. For example, consider the course planning application in Figure 1. Lectures are courses given by a professor or a collaboration partner within a semester for a number of programs. Proposed courses extend lectures by describing which room is requested and which time proposals and which restrictions are made. Planing 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.

$$ProposedCourse \doteq \{ \text{Teacher, Course, Proposal : Kind, Request : Room, Semester, Set2 : \{ Program \}, \} \} \cup \{ \text{Time(Proposal, SideCondition) \}, \Sigma_{ProposedCourse} \} .$$

$$PlannedCourse \doteq \{ \text{ProposedCourse, Reassigned : Kind } \cup \{ \text{Reassigned : Room \} \} \} \cup \{ \text{TimeFrame, TermCourseID \}, \Sigma_{PlannedCourse} \} .$$

$$CourseHeld \doteq \{ \text{PlannedCourse, Reassigned : Room, \{ StartDate, EndDate, AssistedBy \}, \} \} \cup \Sigma_{CourseHeld} \} .$$

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 $R \doteq (R_1, \ldots, R_m, \{B_1, \ldots, B_k\}, \text{id}(R), \Sigma_R)$ is an element of the Cartesian product $R_1^C \times \ldots \times R_m^C \times \text{dom}(B_1) \times \ldots \times \text{dom}(B_k)$. A relationship class $R^C$ consists of a finite set $R^C \subseteq R_1^C \times \ldots \times R_m^C \times \text{dom}(B_1) \times \ldots \times \text{dom}(B_k)$ of objects on $R$ for which $\text{id}(R)$ is a key of $R^C$ and which obeys the constraints $\Sigma_R$.

**Integrity constraints.**

Each database model also uses a set of implicit model-inherent integrity constraints. For instance, relationship types are defined over their component types, and a (relationship) object presumes the existence of corresponding component objects. Typically only finite classes are considered. The EER schema is acyclic. Often names or labels are associated with a minimal semantics that can be derived from the meaning of the words used for names or labels. This minimal semantics allows us to derive synonym, homonym, antonym, troponym, hypernym, and holonym associations among the constructs used.

The most important class of integrity constraints of the EER model is the class of cardinality constraints. Other classes of importance for the EER model are multivalued dependencies, inclusion and exclusion constraints and existence dependencies[7]. Functional dependencies, keys and referential constraints (or key-based inclusion dependencies) can be expressed through cardinality constraints.

Three main kinds of cardinality constraints are distinguished: participation constraints, look-across constraints, and general cardinality constraints. Given a relationship type $R \doteq (\text{compon}(R), \text{attr}(R), \Sigma_R)$, a component $R'$ of $R$, the remaining substructure $R'' = R \setminus R'$ and the remaining substructure $R''' = R'' \cap_R \text{compon}(R)$ without attributes of $R$.

The participation constraint $\text{card}(R, R') = (m, n)$ restricts the number of occurrences of $R'$ objects in the relationship class $R^C$ by the lower bound $m$ and the upper bound $n$. It holds in a relationship class $R^C$ if for any object $o' \in R'^C$ there are at least $m$ and at most $n$ objects $o \in R^C$ with $\pi_R(o) = o'$ for the projection function $\pi_R$ that projects $o$ to its $R'$ components.

Participation constraints relate objects of relationship classes to objects of their component classes. For instance, the constraint $\text{card}(\text{ProposedCourse}, \text{SemesterCourse}) = (0, 3)$ restricts relationship classes for proposals for courses per semester to at least 0 and at most 3, i.e. each course is proposed at 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 $m \leq |\{o \in R^C : \pi_R(o) = o'\}| \leq n$ 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. 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 $R''^C$ may potentially ‘see’ an object $o'$ from $R'^C$. It holds in a relationship class $R^C$ if for any object $o'' \in \text{dom}(R''^C)$ there are at least $m$ and at most $n$ related objects $o'$ with $\pi_R(o) = o'$, i.e. $m \leq |\{o' \in \pi_R(R^C) : o \in R^C \land \pi_R(o) = o' \land o'' \in \text{dom}(R''^C)\}| \leq n$ for any $o'' \in \text{dom}(R''^C)$. 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
combinations. The lower bound expresses that there are Teacher, Room, Kind, and Program which do not have a Docent and Semester combination.

Look-across constraints for a binary relationship type which component types form a key of the relationship type can equivalently expressed by participation constraints, i.e. \( \text{look}(R, R_1) = m_{1..n_1} \) if and only if \( \text{card}(R, R_2) = (m_{1,n_1}) \). Similarly, \( \text{look}(R, R_2) = m_{2..n_2} \) if and only if \( \text{card}(R, R_1) = (m_{2,n_2}) \). This equivalence is neither valid for binary relationship types which cannot be identified by their components and nor for relationship types with more than 2 components.

Participation and look-across constraints can be extended to substructures and intervals and to other types such as entity and cluster types. Given a relationship type \( R \), a substructure \( R' \) of \( R \), \( R'' \) as above. Given furthermore an interval \( I \subseteq \mathbb{N}_0 \) of natural numbers including 0. The (general) cardinality constraint \( \text{card}(R, R') = I \) holds in a relationship class \( R^C \) if for any object \( o' \in \pi_R(R^C) \) there is \( i \in I \) objects \( o \) with \( \pi_R(o) = o' \), i.e. \( |\{o \in R^C : \pi_R(o) = o'\}| \in I \) for any \( o' \in \pi_R(R^C) \).

The following participation, look-across and general cardinality constraints are examples in Figure 1:

- For any \( R' \in \{\text{Semester, Course, Kind}\} \) \( \text{card}(\text{ProposedCourse}, R') = (0, n) \),
- \( \text{card}(\text{ProposedCourse}, \text{Semester Course Teacher}) = (0, 1) \),
- \( \text{card}(\text{CourseHeld, PlannedCourse}) = (1, 1) \),
- \( \text{card}(\text{PlannedCourse, ProposedCourse[Semester] Room TimeFrame}) = (0, 1) \),
- \( \text{card}(\text{ProposedCourse, Docent Semester}) = (0, 3, 4, 5, 6, 7) \).

The first constraint does not restrict the database. The second constraint expresses a key or functional dependency. The types Semester Course Teacher identify any of the other types in the type ProposedCourse, i.e.

\[
\text{ProposedCourse}: \{\text{Semester, Course, Teacher}\} \longrightarrow \{\text{Request, Time, Proposal, Set2}\}
\]

The third constraint requires that any planned course must be given. The fourth constraint requires that rooms are not overbooked. The fifth constraint allows that docents may not teach in a semester, i.e. have a sabbatical. If a docent is teaching in a semester then at least 3 and at most 7 courses are given by the docent.

Look-across constraints were originally introduced by P.P. Chen [1] as cardinality constraints. UML uses look-across constraints. Participation and look-across constraints cannot be axiomatised through a Hilbert- or Gentzen-type logical calculus. If only upper bounds are of interest then an axiomatisation can be found in [3] and [4]. General cardinality constraints combine equality-generating and object-generating constraints such as keys, functional dependencies and referential integrity constraints into a singleton construct.

Logical operators can be defined for each type. A set of logical formulas using these operators can define the integrity constraints which are valid for each object of the type.

Schemata.

The schema is based on a set of base (data) types which are used as value types for attribute types. A set \( \{E_1, ..., E_n, C_1, ..., C_l, R_1, ..., R_m\} \) 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 \( D = (S, \Sigma) \) where \( S \) is a schema and \( \Sigma \) is a set of constraints. A database \( D^C \) on \( D \) consists of classes for each type in \( D \) such that the constraints \( \Sigma \) are valid.

The classes of the extended ER model have been defined through sets of objects on the types. In addition to sets, lists, multi-sets or other collections of objects may be used. In this case, the definitions used above can easily be extended [8].

A number of domain-specific extensions have been introduced to the ER model. One of the most important is the extension of the base types by spatial data types such as: point, line, oriented line, surface, complex surface, oriented surface, line bunch, and surface bunch. These types are supported by a large variety of functions such as: meets, intersects, overlaps, contains, adjacent, planar operations, and a variety of equality predicates.

The translation of the schema to (object-)relational or XML schemata can be based on a profile [8]. Profiles define which translation choice is preferred over other choices, how hierarchies are treated, which redundancy and null-value support must be provided, which kind of constraint enforcement is preferred, which naming conventions are chosen, which alternative for representation of complex attributes is preferred for which types, and whether weak types can be used. The treatment of optional components is also specified through the translation profile of the types of the schema. A profile may require the introduction of identifier types and base the identification on the identifier. Attribute types may be translated into data formats that are supported by the target system.
The EER schema can be used to define views. The generic functions insert, delete, update, projection, union, join, selection and renaming can be defined in a way similarly to the relational model. Additionally, nesting and unnesting functions are used. These functions form the algebra of functions of the schema and are the basis for defining queries. A singleton view is defined by a query that maps the EER schema to new types. Combined views also may be considered which consist of singleton views which together form another EER schema. A view schema is specified over an EER schema \( D \) by a schema \( V = \{ S_1, ..., S_m \} \), 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 \( \text{card}(R, R') = (m, n) \) are used for component consisting of one type \( R' \) then the arc from \( R \) to \( R' \) is labelled by \( (m, n) \). If look-across constraints \( \text{look}(R, R') = m..n \) are used for binary relationship types then the arc from \( R \) to \( R' \) is labelled by \( m..n \).

**KEY APPLICATIONS**
The main application area for extended ER models is the conceptualisation of database applications. Database schemata can be translated to relational, XML or other schemata based on transformation profiles that incorporate properties of the target systems.

**FUTURE DIRECTIONS**
The ER model has had a deep impact on the development of diagramming techniques in the past and is still influencing extensions of the unified modelling language UML. UML started with binary relationship types with look-across constraints and without relationship type attributes. Class diagrams currently allow n-ary relationship types with attributes. Relationship types may be layered. Cluster types and unary relationship types allow for distinguishing generalisation from specialisation. ER models are not supported by native database management systems and are mainly used for modelling of applications at the conceptual or requirements level. ER schemata are translated to logical models such as XML schemata or relational schemata or object-relational schemata. Some of the specifics of the target models are not well supported by ER models and must be added after translating ER schemata to target schemata, e.g., specific type semantics such as list semantics (XML) or as special ordering or aggregation treatment of online analytical processing (OLAP) applications. The ER model has attracted a lot of research over the last 30 years. Due to novel applications and to evolution of technology old problems and novel problems are challenging the research on this model. Typical old problems that are still not solved in a satisfactory manner are: development of a science of modelling, quality of ER schemata, consistent refinement of schemata, complex constraints, normalisation of ER schemata, normalisation of schemata in the presence of incomplete constraint sets. Novel topics for ER research are for instance: evolving schema architectures, collaboration of databases based on collaboration schemata, layered information systems.
and their structuring, schemata with redundant types, ER schemata for OLAP applications. Structures of database applications are often represented through ER models. Due to the complexity of applications, a large number of extensions have recently been proposed, e.g., temporal data types, spatial data types, OLAP types and stream types. Additionally, database applications must be integrated and cooperate in a consistent form. The harmonisation of extensions and the integration of schemata is therefore a never ending task for database research.

ER models are currently extended for support of (web) content management that is based on structuring of data, on aggregation of data, on extending data by concepts and on annotating data sets for simple reference and usage. These applications require novel modelling facilities and separation of syntactical, semantical and pragmatic issues. The ER model can be extended to cope with these applications.

The ER model is mainly used for conceptual specification of database structuring. It can be enhanced by operations and a query algebra. Operations and the queries can also be displayed in a graphical form, e.g. on the basis of VisualSQL. Most tools supporting ER models do not currently use this option. Enhancement of ER models by functionality is necessary if the conceptualisation is used for database development. Based on functionality enhancement, view management facilities can easily be incorporated into these tools.

ER models are becoming a basis for workflow systems data. The standards that have been developed for the specification of workflows have not yet been integrated into sophisticated data and application management tools.

URL TO CODE
http://www.informatik.uni-kiel.de/~thalheim/HERM.htm
http://www.is.informatik.uni-kiel.de/~thalheim/indeeerm.htm

Readings on the RADD project (Rapid Application and Database Development)

CROSS REFERENCE
I. DATABASE FUNDAMENTALS
   a. Data models (including semantic data models)
   b. Entity-Relationship (ER) model
   c. Unified modelling language (UML)

III. THEORETICAL ASPECTS
   b. Relational Theory

RECOMMENDED READING
Between 3 and 15 citations to important literature, e.g., in journals, conference proceedings, and websites.

Specialisation and Generalisation

Bernhard Thalheim

Christian-Albrechts University Kiel, http://www.informatik.uni-kiel.de/~thalheim/HERM.htm

SYNONYMS
refinement, abstraction, hierarchies;
clustering, grouping, inheritance

DEFINITION
Generalisation and specialisation are main principles of database modelling. Generalisation maps or groups types or classes to more abstract or combined ones. It is used to combine common features, attributes, or methods. Specialisation is based on a refinement of types or classes to more specific ones. It allows developers to avoid null values and to hide details from non-authorised users. Typically, generalisations and specialisations form a hierarchy of types and classes. The more specialised classes may inherit attributes and methods from more general ones. In database modelling and implementation clusters of types to a type that represents common properties and abstractions from a type are the main kinds of generalisations. Is-A associations that specialise a type to a more specific one and Is-A-Role-Of associations that considers a specific behaviour of objects are the main kinds of specialisations.

MAIN TEXT
Specialisation introduces a new entity type by adding specific properties belonging to that type which are different from the general properties of its more general type. Generalisation introduces the Role-Of relationship or the Is-A relationship between a subtype and its general type. Therefore, the application, implementation, and processes are different. For generalisation the general type must be the union of its subtypes. The subtypes can be virtually clustered by the general type. This tends not to be the case for specialisation. Specialisation is a refinement or restriction of a type to more special ones. Typical specialisations are Is-A and Has-Role associations. Exceptions can be modelled by specialisations.

Different kinds of specialisation may be distinguished: structural specialisation which extends the structure, semantic specialisation which strengthens type restrictions, pragmatical specialisation which allows to separate the different usage of objects in contexts, operational specialisation which introduces additional operations, and hybrid specialisations. Is-A specialisation requires structural and strong semantic specialisation. Is-A-Role-Of specialisation requires structural, pragmatical and strong semantic specialisation.

Generalisation is based either on abstraction or on grouping. The cluster construct of the extended ER model is used to represent generalisations. Generalisation tends to be an abstraction in which a more general type is defined by extracting common properties of one or more types while suppressing the differences between them. These types are subtypes of the generic type. New types are created by generalizing classes that already exist. Structural combination typically assumes the existence of a unifiable identification of all types. Semantical combination allows the disjunction of types through the linear sum of semantics. Pragmatical generalisation is based on building collections whenever applications require a consideration of commonalties.

CROSS REFERENCE
I. DATABASE FUNDAMENTALS
   a. Data models (including semantic data models)

REFERENCES
Abstraction

Bernhard Thalheim
Christian-Albrechts University Kiel, http://www.informatik.uni-kiel.de/~thalheim/HERM.htm

SYNONYMS
component abstraction, localisation abstraction, implementation abstraction;
association, aggregation, composition, grouping, specialisation, generalisation, classification

DEFINITION
Abstraction allows developers to concentrate on the essential, relevant or important parts of an application. It uses a mapping to a model from things in reality or from virtual things. The model has the truncation property, i.e. it lacks some of the details in the original, and a pragmatic property, i.e. the model use is only justified for particular model users, tools of investigation, and periods of time. Database engineering uses construction abstraction, context abstraction and refinement abstraction. Construction abstraction is based on the principles of hierarchical structuring, constructor composition, and generalisation. Construction abstraction assumes that the surroundings of a concept are commonly assumed by a community or within a culture and focuses on the concept, turning away attention from its surroundings such as the environment and setting. Refinement abstraction uses the principle of modularisation and information hiding. Developers typically use conceptual models or languages for representing and conceptualising abstractions. The enhanced entity-relationship model schema are typically depicted by an EER diagram.

MAIN TEXT
Database engineering distinguishes three kinds of abstraction: construction abstraction, context abstraction and refinement abstraction.
Constructor composition depends on the constructors as originally introduced by J. M. Smith and D.C.W. Smith. Composition constructors must be well founded and their semantics must be derivable by inductive construction. There are three main methods for construction: development of ordered structures on the basis of hierarchies, construction by combination or association, and construction by classification into groups or collections. The set constructors \( \subset \) (subset), \( \times \) (product) and \( \mathcal{P} \) (powerset) for subset, product and nesting are complete for the construction of sets.
Subset constructors support hierarchies of object sets in which one set of objects is a subset of some other set of objects. Subset hierarchies are usually a rooted tree. Product constructors support associations between object sets. The schema is decomposed into object sets related to each other by association or relationship types. Power set constructors support a classification of object sets into clusters or groups of sets - typically according to their properties.
Context abstraction allows developers to commonly concentrate on those parts of an application that are essential for some viewpoints during development and deployment of systems. Typical kinds of context abstraction are component abstraction, separation of concern, interaction abstraction, summarisation, scoping, and focusing on typical application cases. Component abstraction factors out repeating, shared or local patterns of components or functions from individual concepts. It allows developers to concentrate on structural or behavioral aspects of similar elements of components. Separation of concern allows developers to concentrate on those concepts that are a matter of development and to neglect all other concepts that are stable or not under consideration. Interaction abstraction allows developers to concentrate on those parts of the model that are essential for interaction with other systems or users. Summarisation maps the conceptualisations within the scope to more abstract concepts. Scoping is typically used to select those concepts that are necessary for current development and removes those concepts that do not have an impact on the necessary concepts. Database models may cover a large variety of different application cases. Some of them reflect exceptional,
abnormal, infrequent and untypical application situations. Focusing on typical application cases explicitly separates models for the normal or typical application case from those that are atypical. Atypical application cases are not neglected but can be folded into the model whenever atypical situations are considered.

The context abstraction concept is the main concept behind federated databases. Context of databases can be characterized by schemata, version, time, and security requirements. Sub-schemata, types of the schemata or views on the schemata, are associated by explicit import/export bindings based on a name space. Parametrisation lets developers to consider collections of objects. Objects are identifiable under certain assumptions and completely identifiable after instantiation of all parameters.

Interaction abstraction allows developers to display the same set of objects in different forms. The view concept supports this visibility concept. Data is abstracted and displayed in various levels of granularity. Summarisation abstraction allows developers to abstract from details that are irrelevant at a certain step. Scope abstraction allows developers to concentrate on a number of aspects. Names or aliases can be multiply used with varying structure, functionality and semantics.

Refinement abstraction is mainly about implementation and modularisation. It allows developers to selectively retain information about structures. Refinement abstraction is defined on the basis of the development cycle (refinement of implementations). It refines, summarises and views conceptualizations, hides or encapsulates details or manages collections of versions. Each refinement step transforms a schema to a schema of finer granularity. Refinement abstraction may be modelled by refinement theory and infomorphisms. Encapsulation removes internal aspects and concentrates on interface components. Blackbox or graybox approaches hide all aspects of the objects under consideration. Partial visibility may be supported by modularisation concepts. Hiding supports differentiation of concepts into public, private (with the possibility to be visible to ‘friends’) and protected (with visibility to subconcepts). It is possible to define a number of visibility conceptualizations based in inflection. Inflection is used for the injection of combinable views into the given view, for tailoring, ordering and restructuring of views, and for enhancement of views by database functionality. Behavioral transparency is supported by the glassbox approach. Security views are based on hiding. Versioning allows developers to manage a number of concepts which can be considered to be versions of each other.

CROSS REFERENCE
I. DATABASE FUNDAMENTALS
   a. Entity-Relationship Model, Extended Entity-Relationship Model, Object Data Models, Object Role Modeling, Unified Modeling Language

REFERENCES