INSTITUT FÜR INFORMATIK

An Approach to Conceptual Schema Evolution

Gunar Fiedler, Bernhard Thalheim

Bericht Nr. 0701
January 2007

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

An Approach to Conceptual Schema Evolution

Gunar Fiedler, Bernhard Thalheim

Bericht Nr. 0701
January 2007

e-mail: {fiedler,thalheim}@is.informatik.uni-kiel.de

Abstract

In this work we will analyse conceptual foundations of user centric content management. Content management often involves integration of content that was created from different points of view. Current modeling techniques and especially current systems lack of a sufficient support of handling these situations. Although schema integration is undecideable in general, we will introduce a conceptual model together with a modeling and maintenance methodology that simplifies content integration in many practical situations. We will define a conceptual model based on the Higher-Order Entity Relationship Model that combines advantages of schema oriented modeling techniques like ER modeling with element driven paradims like approaches for semistructured data management. This model is ready to support contextual reasoning based on local model semantics. For the special case of schema evolution based on schema versioning we will derive the compatibility relation between local models by tracking dependencies of schema revisions. Additionally, we will discuss implementational facets, such as storage aspects for structurally flexible content or generation of adaptive user interfaces based on a conceptual interaction model.

1 Introduction

1.1 Organizational Aspects of Mid-Ranged Information Systems

Information systems support a wide range of application scenarios. Information system research usually pays attention to "very large" systems because classical problems in the area of databases and information retrieval are trivial for small amounts of data but scale poorly for large or very large systems.
Classical very large scaled information systems are usually equipped with a large amount of technical and personal resources. Systems are designed carefully, usually in a handicrafted way by database modeling experts using different kinds of (potentially expensive) case tools. Becoming productive, systems are supervised and tuned by administrators while developers prepare and create new versions to make the system ready for new challenges.
Nowadays, the technical infrastructure is cheap enough to use information systems technologies in many different application scenarios beside the classical way. Creation, deployment, and administration of such systems became easy and self-* technologies enable "out of box" usage. That's why databases as well as other information systems technologies can be found at any scale: as support for personal homepages as well as big business intelligence applications.
Usage of these technologies at smaller scale significantly differs from the "very large scaled" systems described above: usually, there is no real organizational and personal infrastructure the systems can rely on. Automation of central tasks is not only a feature but simply necessary to make the system work. While automation of administrative tasks is widely available, the process of system design, creation, and deployment needs additional computational as well as organizational support.
This work considers mid-sized information systems. These systems are usually responsible to serve within a specialized application context. Within this context these information systems play a central role and heavily support processes and decisions. As a major characteristics these systems are usually bound to a very limited budget with possibly changing investors. Like any other information system these applications are designed for long-term usage. While the main tasks of these systems remain stable, the focus changes over time and brings new challenges.
Because of the strictly limited budget these systems are usually developed and administrated by only a small part time staff. These people are usually domain experts but the database knowledge comes from textbooks and some practical experience in using databases. Modeling knowledge and advanced programming skills are usually

not present. These people are system users, developers, and administrators in one person. Additionally, personal changes occur very frequently. Because of missing knowledge management facilities, every loss of employees implies a loss of knowledge about the system and the data.
To limit our focus at the other side, we nevertheless concentrate on systems with an existing and at least partially known domain model. Domain experts with some computer science skills are available that are able to formulate the needs and wishes of the system's properties and behaviour in a structured way.
Taking this situation as the present state which cannot be changed, the information system itself has to adapt. In this work we will analyse aspects of making an information system robust against rework and changing requirements due to limited modeling resources at any stage of the system's lifecycle. Automation of central aspects in system delevopment and deployment is seen as a major step towards releasing domain experts from technical details and therefore potential errors and risks.

1.2 Combining Content Management Systems and Data Warehouses

In this work we look at applications which combine in their requirements portfolio aspects of two major streams in information systems technologies ([FCF+06]):

- *Content Management Systems* are used for the collaborated creation, processing, usage, and deployment of content. Content management systems are mostly (but not exclusivly) used for deployment of semistructural data like (hyper-)texts and multimedia via multiple channels like HTML pages, print media, or mobile devices.

- *Data Warehouses* store huge amounts of (usually equally and simply structured) data. They facilitate various data analysis and reporting tools as a base for decision support and data mining activities.

Applications in our focus facilitate both sides:

- They work as archives for valueable data. This data is the base for decision and process support. Depending on current requirements different kinds of data analysis, reporting, and mining have to be facilitated.

- Due to the organizational limitations described above, a fully integrated solution like in classical data warehouse scenarios is not achievable. Rework on the system, changing requirements, and changing personal with different abilities cause heterogenous but partially overlapping data structures and processes. Querying and reporting should be possible beside these shortcomings.

- Typical content management functionality like automated generation of reports, forms, and presentations has to be integrated, too.
- Beside these building blocks additional aspects become relevant: to prevent loss of knowledge within the developer's community, knowledge management has to be supported. The base ground for a solid and partially automated knowledge management is the availability of an explicit model of the application's domain. Design decisions from the modeling process have to be tracked to enable forthcoming developers to understand the system as well as to support automated validation of ensured system properties during redesign activities.
- Domain experts with lacking technical experiences have to be supported by an appropriate system abstraction.

In the following we will call such systems *content warehouses*.

1.3 Schema Evolution Case Study: A Data Warehouse for Questionnaires

As an example of a mid-range information system with the described characteristics we will introduce the wild animal cadaster maintained by the Hunting Association of Schleswig-Holstein. The wild animal cadaster consists of a database which stores biological and ecological facts about different species in the context of hunting and forest care. The data is raised by evaluating questionnaires that were filled by volunteers among hunters and tenants as well as professional biologists. Questionnaires focus on certain species and associated aspects like counting foxes or partridges. Questionnaires are repeated regularily but possibly with changing questions. All questionnaires are collected within wild animal cadaster. This data is the base for decisions about protection of certain species, hunting quotas, and compensations for farmers. Beside this direct usage the data can be used for biological and ecological research in a wide sense, e.g. the relationships between species or the influence of agricultural usage on the presence of species.
The wild animal cadaster will be used as an example within this work in the following way:

- There is a heterogenous, partially matching data schema which changes over time (the project already lasts for more than 10 years.)
- Queries and reports use the full stock of data, so there is a need for integration and change propagation.
- The application itself is not only an isolated database but it facilitates content management scenarios: questionnaires must be created, the data aquisition must work synchronously.

- The argumentation for knowledge management and technical abstraction applies: design decisions, e.g. dependencies and relationships between questionnaires have to be explicitly tracked and the application must be self-descriptive for documentation and development purposes.

1.4 The Need for Schema Evolution

The argumentation for schema evolution bases on pure pragmatical needs. But is there any argument beside "correcting errors" that supports design strategies facilitating local design decisions?

Phenomenas in real world exist independently from one's perception of the world. But: is there a unique reality? We cannot answer this question, because the only way to access the "facts" of the world is observation of perceivable phenomenas. These observations are subjective for every single participating observer. Analyzing the way of percepting the world and the discovery of new facts during observation leads naturally to the assumption that observation is never complete. We have to be aware of the fact that there are phenomenas and causal relationships between phenomenas that are hidden from our perception. These phenomenas may or may not be visible to other observers in a way we would also see or in a way which is unaccessable for our observer at all. Our picture of the world may depend on our current mental state, too. In different situations we may consider different intensions about reality: we consider different aspects as important or we hide parts of our known part of reality.
For that reason our representation has to separate between real world phenomenas and intensions as observations of the world. But this argumentation lacks of consistency: how we should model phenomenas independently from one's observation if we can only access these phenomenas by observation? To overcome this problem we can introduce all-embracing observations which include every phenomena that was observed at any time by any known observer. This all-embracing observations represent the amount of knowledge about the world in a certain situation. We have to be aware that this knowledge will change over time: when new observers join or new phenomenas are discovered. On a general point of view we call real world phenomena *objective* if they are part of every available observation. All other phenomena are called *subjective*.

2 Conceptual Models of Content Management Systems

2.1 Separation of Concerns

Classical conceptual models of database systems express the structural properties of the application domain that is manifested within this system. Content management systems add new architectural aspects.
Currently, many systems claim to be content management systems. 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 generation, 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.
Content is characterized by three dimensions: the underlying raw data enhanced by some meta information, the structuring of media objects which is actually a suite of views over the raw data, and the presentation, consisting of stylesheet suites. These aspects can be found in any slightly generic CMS while the implementations differ from one system to another. For example, structuring may be definable as suites of relational views or as an object-oriented document model. Analogously, the presentation of content is usually implemented as a set of templates or stylesheets, e.g. in a scripting language.
Although the need for separation between these dimensions is very present, available systems lack of a conceptual and computational support for more sophisticated tasks. Structuring as well as presentation of content strongly depends on the contents usage, it depends on the users profile and task portfolio. Different users need different "information units" to perform their associated tasks. These information units may have different or multiple structurings because they may be stored in or delivered by different independent sources. Analogously, these units have to be made accessible to the user with respect to her abilities and needs.
All this can be done using a traditional CMS by coding the logic to the view suite or presentation templates. But following the tradition of database management systems this functionality should be provided by the CMS in a more generic way. To support reuse of information units in different situations and under different circumstances an explicit representation of the concepts behind the data is necessary. The metadata approach of today's content management systems only facilitates

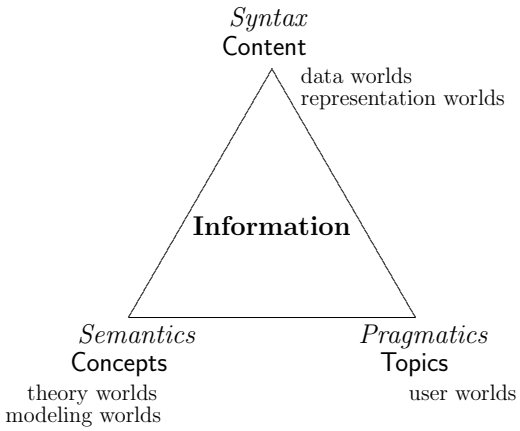

Figure 2.1: Semiotic Separation of Concerns

manual or semiautomatical reuse strategies. A similar discussion occurs for handling presentation of information unis. There is not only a need for templates at late abstraction layers but a conceptual representation of how presentation takes place.

To define a conceptual model for content management systems, the different aspects mentioned above should be separated to allow a dedicated and understandable handling during system design. [FT04] proposes a conceptual model as depicted in figure 2.1 consisting of three pairwise dependent components: handling of structure ("'content"'), handling of semantics ("'concepts"') and handling of usage ("'topics"'). In the following we will deepen these three dimensions to analyse the situation in our context and to develop a conceptual model that describes the content management system according to this context.

2.2 The Content Part: Describing Structure

The structural part looks at the universe of discourse by describing intensions about the nature of "objects" within this universe. According to Peirce's Theory of Signs ([Pei58]) there exist three phenomenological categories or 'modes of being' that can be used to organize all human experience:

- The *Firstness* category studies phenomenas as objects of perception. It considers that which is without reference to any other one, such as qualities or feelings.
- The *Secondness* category studies proper relations between phenomena. Examples are action, reaction, causality, or factuality.
- The *Thirdness* category brings Firstness and Secondness into relation with each other. It may be manifested by representation, thought, or generality.

To express these categories, the conceptual model presented here uses the notions of *entites*, *properties*, *relationships*, and (structural) *concepts*. Two different approaches for defining a structural conceptual model can be distinguished:

- *Type driven:* the conceptual model defines a set of types like entity types and relationship types in the well-known Entity Relationship Model, or classes in UML. The state of a database valid for a such a schema is a set of objects (respectivly entities and relationships) fulfilling the integrity constraints induced by these types. These languages are often guarded by a closed world assumption: aspects, that are not modeled by a type, do not exist. For example, an entity owns exactly the attributes that are defined in its corresponding entity type[1].

- *Element driven:* the conceptual model is based on the assumption, that the elements under consideration (objects, entities, relationships, etc.) exist independently from one's perception and independently from their representation in a model. Using this point of view the conceptual model does not define types, but individual representations of the element's characteristics. Based on these characteristics valid schemata are derived or induced as a comprehension or intensional description of the elements. This approach is widely used in the area of semistructured data representations: semistructured elements are self-descriptive data structures, usually represented by a (more or less general purpose) graph model, see e.g. the OEM data model ([MAG+97]) or the document object model for XML data ([W3C04a]). Based on collections of such elements schemata can be defined: [GW97] introduces the notion of *DataGuides* as a comprehension of OEM graphs. XML documents can be validated against schema definitions provided by DTDs or XML Schema Documents ([W3C04b]). Tool support for extracting XML schema definitions or DTDs is widely available. The element driven approach is per definition based on an open world assumption. This is also reflected in design guidelines e.g. for XML schemata: the language explicitly supports declarations for *any element* and *any attribute* with the possibility to completely skip this content.

Type driven schema design is widely used, because strict typing offers great potential for optimizations on subsequent system layers. It forces the designer to create a well-thought design, but the result is often inflexible in the case of changing requirements. Pure element driven models are the most flexible solutions, but unmanaged changes usually lead to uncontrollable data structures with the full spectrum of integration problems, see e.g. [BLN86].
The model defined in the following discussion is element driven to provide flexibility for integration, but also uses the aspects of type driven modeling (in particular the

[1] If you consider e.g. IS-A hierarchies in ERMs or inheritance in UML, this statement has to be relaxed, because an entity can be member of several entity type extensions. But nevertheless, there will always be a most specific entity type that models the entity completely.

Higher-Order Entity Relationship Model defined in [Tha00]) whenever it is meaningful.

To manifest objects on the conceptual layer, the notion of *entities* is introduced: an entity is considered to be distinguishable from other entities. It exists independantly from other entities. By now, it is defined as an abstract element from a given universe $\mathcal{U}^E$.
Given a universe of roles $\mathcal{U}^R$ we can define the notion of relationships. A first order relationship is a partial injective function $\mathcal{U}^R \longrightarrow \mathcal{U}^E$ that assigns roles to entities. Entities $e_1, ..., e_n$ are related to each other if there is a relationship r that assigns a role r_i to the entity e_i.
Based on the definition of first order relationships we can introduce higher order relationships. Entities are considered to be relationships of order 0. A relationship of order $i > 0$ is a partial function that maps roles to relationships $r_1, ..., r_n$ with $max(order(r_1), ..., order(r_n)) = i - 1$. In the following, we will denote the set of all relationships of order i with $\mathcal{R}^i$. $\mathcal{R}^* = \bigcup_{i=0}^{\infty} \mathcal{R}^i$ will denote the set of all possible relationships.
Consider distinct sets of values $\mathcal{D}_i$ (base domains) and a universe $\mathcal{U}^V = \bigcup_i \mathcal{D}_i$ of values. A property is defined as a partial, surjective function $p : \mathcal{R}^* \longrightarrow \mathcal{D}_i$ for an index i that assigns values from $\mathcal{D}_i$ to relationships from $\mathcal{R}^*$. This definition naturally includes properties of entities. As usual, the domain of the function p will be denoted by $dom(p)$. The definition enforces distinct base domains $\mathcal{D}_i$ for property domains to prevent implicit semantic relationships between values of different domains. Domains should be designed to fit the set of values in real world as perfect as possible. For that reason, each domain should be the smallest set of values that contains all necessary values. To support (or to force) the designer not to use "super domains" like `string` the definition calls for surjective domain functions: each element in the chosen base domain $\mathcal{D}_i$ is possibly necessary.
Structural concepts are bringing together Firstness and Secondness. There are two different ways to look at concepts:

- The *extensional* point of view describes a concept as a set of elements.
- The *intensional* point of view reveals the characteristics of the concept. It describes properties, relationships, and constraints typical for this concept.

Intension and extension of a concept are strongly related: every element that is member of the concept's extension satisfies the intensional constraints; especially, it owns the properties and relationships defined in the concept's intension. On the other hand, the concept's intension describes exactly the concept's extension: there is no element, that satisfies the intensional constraints, but is not a member of the concept's extension.
We consider a universe $\mathcal{U}^{\mathfrak{C}}$ of abstract concepts. Expressing concepts is realized by partial concept functions $\mathcal{I}^0$, $\mathcal{E}^0$:

- The (initial) intension function $\mathcal{I}^0 : \mathcal{U}^{\mathfrak{C}} \longrightarrow 2^{(\mathcal{L},\mathcal{F})}$ maps abstract concepts to tuples $(\mathcal{L}, \mathcal{F})$ where $\mathcal{L}$ is a language and $\mathcal{F}$ is a set of sentences of $\mathcal{L}$ expressing intensional constraints. At this point we do not restrict the language $\mathcal{L}$ with the exception that $\mathcal{L}$ should allow to express mandatory, possible, and forbidden properties and relationships for the elements in the concept's extension. Other constraints may include for example key or cardinality constraints, generalization / specialization of concepts, union types, and others.
- The (initial) extension function $\mathcal{E}^0 : \mathcal{U}^{\mathfrak{C}} \longrightarrow 2^{\mathcal{R}^*}$ maps abstract concepts to sets of relationships (including entities).

For a given universe $\mathcal{U}^N$ of names we define a partial concept naming function $\mathcal{N} : \mathcal{U}^{\mathfrak{C}} \longrightarrow \mathcal{U}^N$ that assigns names to some concepts from $\mathcal{U}^{\mathfrak{C}}$. Concepts $\mathfrak{C}$ where $\mathcal{N}(\mathfrak{C})$ is defined are called *named concepts*.
Based on the initial concept functions, four different types of concepts can be distinguished:

1. Concepts $\mathfrak{C} \in \mathcal{U}^{\mathfrak{C}}$ where $\mathcal{I}^0(\mathfrak{C})$ as well as $\mathcal{E}^0(\mathfrak{C})$ are undefined are called *pure abstract concepts*.
2. Concepts $\mathfrak{C} \in \mathcal{U}^{\mathfrak{C}}$ where $\mathcal{I}^0(\mathfrak{C})$ is defined but $\mathcal{E}^0(\mathfrak{C})$ is undefined are called *intension driven concepts*.
3. Concepts $\mathfrak{C} \in \mathcal{U}^{\mathfrak{C}}$ where $\mathcal{E}^0(\mathfrak{C})$ is defined but $\mathcal{I}^0(\mathfrak{C})$ is undefined are called *assertion driven concepts*.
4. Concepts $\mathfrak{C} \in \mathcal{U}^{\mathfrak{C}}$ where $\mathcal{I}^0(\mathfrak{C})$ as well as $\mathcal{E}^0(\mathfrak{C})$ are defined are called *constraint driven concepts*.

The initial concept functions $\mathcal{I}^0$ and $\mathcal{E}^0$ determine intension and extension of concepts:

1. *pure abstract concepts:* Intension and extension are unknown in an open world sense.
2. *intension driven concepts:* $\mathcal{I}(\mathfrak{C}) = \mathcal{I}^0(\mathfrak{C})$, $\mathcal{E}(\mathfrak{C}) = \{r \in \mathcal{R}^* | r \models_{\mathcal{L}} \mathcal{F}\}$
3. *assertion driven concepts:* $\mathcal{E}(\mathfrak{C}) = \mathcal{E}^0(\mathfrak{C})$. The intension $(\mathcal{L}, \mathcal{F})$ of an assertion driven concept can only be formulated modulo a given language $\mathcal{L}$. As a necessary condition, $\mathcal{F}$ is a set of sentences $a \in \mathcal{L}$ such that

$$(\forall r \in \mathcal{E}(\mathfrak{C}))(r \models_{\mathcal{L}} \mathcal{F}) \quad \wedge \quad (\forall \tilde{r} \notin \mathcal{E}(\mathfrak{C}))(\tilde{r} \not\models_{\mathcal{L}} \mathcal{F})$$

   This condition will not fully determine the concept's intension, but from a pragmatical point of view it is sufficient to pick an intension from the set of possible intensions based on metrics, e.g.

- "smaller" theories (number of sentences and base terms) are prefered
- theories relating the intension to named concepts are prefered
- Given a "seed" intension $\mathcal{F}^0$, theories are prefered that keep as close as possible to $\mathcal{F}^0$.
- An indeterministic choice may also be appropriate in certain situations.

Assuming that the state of the system will not remain stable over time (e.g. updates occur), estimations of intensions may be considered, e.g.

- *best current explaination:* together with a metrics the necessary condition is relaxed to
$$(\forall r \in \mathcal{E}(\mathfrak{C}))(r \models_{\mathcal{L}} \mathcal{F})$$
- *sufficient explaination:* The "best current explaination" is relaxed by a threshold value V that allows a number of estimation errors:
$$|\{r \in \mathcal{E}(\mathfrak{C}) | r \not\models_{\mathcal{L}} \mathcal{F}\}| \leq V$$
- *best oracle:* Given a time horizon t_{max} and a function $\delta^{\mathfrak{C}} : T \longrightarrow \mathcal{U}^{\mathfrak{C}}$ that describes the changes of concept $\mathfrak{C}$ over time[2], we choose the theory that will be "best current explaination" for most points in time $t \leq t_{max}$.

Picking the intension for a given extension will be abbreviated by the "explain" operator $\upharpoonleft \mathfrak{C} = \upharpoonleft \mathcal{E}(\mathfrak{C})$ in the following discussion. Similarly, we define the "sample" operator $\downharpoonleft \mathfrak{C} = \downharpoonleft \mathcal{I}(\mathfrak{C})$.

$$\downharpoonleft \mathcal{I}(\mathfrak{C}) = \begin{cases} \{r \in \mathcal{R}^* | r \models_{\mathcal{L}} \mathcal{F}\} & , \mathcal{I}(\mathfrak{C}) \; is \; defined \\ undefined & , \mathcal{I}(\mathfrak{C}) \; is \; undefined \end{cases}$$

4. *constraint driven concepts:* Assuming exact and deterministic explain operators we can define
$$\begin{aligned} \mathcal{I}(\mathfrak{C}) &= \mathcal{I}^0(\mathfrak{C}) \sqcup \upharpoonleft \mathcal{E}^0(\mathfrak{C}) \\ \mathcal{E}(\mathfrak{C}) &= \mathcal{E}^0(\mathfrak{C}) \cup \downharpoonleft \mathcal{I}^0(\mathfrak{C}) \end{aligned}$$

   In any other case we have to consider the general definition given by a mutual recursion:

[2] We consider discrete points of time.

$$\begin{aligned}
\mathcal{I}^k(\mathfrak{C}) &= \mathcal{I}^{k-1}(\mathfrak{C}) \sqcup \nwarrow\!\!\!\!\nearrow \mathcal{E}^{k-1}(\mathfrak{C}) \\
\mathcal{E}^k(\mathfrak{C}) &= \mathcal{E}^{k-1}(\mathfrak{C}) \cup \swarrow\!\!\!\!\searrow \mathcal{I}^{k-1}(\mathfrak{C}) \\
\mathcal{I}(\mathfrak{C}) &= \mathcal{I}^{\infty}(\mathfrak{C}) \\
\mathcal{E}(\mathfrak{C}) &= \mathcal{E}^{\infty}(\mathfrak{C})
\end{aligned}$$

A conceptual schema is a set of concepts $\{\mathfrak{C}_1, ..., \mathfrak{C}_n\}$.

2.3 Intensions as Mental Models of the World

Intensional logic attempts to study entites without extensionality ([Bea98]) such as qualities, attributes, properties, relations, conditions, states, concepts, ideas, notions, propositions, and thoughts. It distinguishes between the meaning of a term and the entity the term designates ([Fit06]).
To handle intensional entites a possible world reduction to extensionality may be used. In [Fit04] a two-sorted modal logic approach is introduced. Tichý's Transparent Intensional Logic (TIL, see e.g. [DM00]) defines intensional entities as functions over time and modality which allow the construction of expressions' intensions based on the intensions of subexpressions. We will follow this approach in a slightly modified way.
The base for intensions are atomic types α_i of elements, at least

- a set of truth values denoted by o, and
- distinct sets of individuals denoted by ι_i. Sets of individuals and truth values are distinct, too.

Tichý introduced a single class ι of individuals. In the following discussion we will refer to a single class ι of individuals, but we allow arbitrary classes ι_i in general as long as these classes are kept fixed throughout the whole discussion.
To express possible worlds for our intensional entities we need additional sets of modalities. Tichý restricted his attention to two modalities ω (the set of possible worlds) and τ (the set of numbers representing time lines for a possible world). We generalize this notion to an arbitrary multi-modal approach with a fixed set $\Omega = \{\omega_1, ..., \omega_n\}$ of sets for modalities. Typical modalities are:

- Possible instances of the concept, e.g. objects in object oriented models or entities and relationships in entity relationship models, see also [Fit00]. For example, if we consider the intension behind a person we can look at a single person as a possible world. In the following discussion we will refer to this modality by the set ω which represents possible instances. A single possible instance from ω will be denoted by w (compared to "object identifiers" in object orientation.)

- Time, we will denote the (ordered) set of time points by τ and a single point of time by t.
- Partial knowledge or variations: in some cases we have only partial knowledge about the world. This results in alternative representations e.g. for instances. Alternatives may be represented as different instances or as variations of the same instance. The second case implies another modality μ with a variation $m \in \mu$.

The choice of modalities is arbitrary, in examples we will refer to the introduced modalities ω (instances), τ (time), μ (alternative choice).
Over the objectual base $\{o, \iota_i, \omega_j\}$ we can define first order types:

1. o, ι_i, ω_j are first order types.
2. All partial functions $f : \alpha_1 \times ... \times \alpha_n \rightarrow \beta$ with first order types α_i and β are first order types, too. In the following we use the Polish notation of Tichý: $f/\beta\alpha_1...\alpha_n$ denotes the function f from above.
3. Nothing else is a first order type.

We assume standard base functions to be defined like connectives ($\bot$, $\top$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, ...) for truth values, equivalence for nominal individuals and modalities, ordering relations for ordinal individuals and modalities (e.g. time modalities) and so on.
Based on this definition of types we can adapt Tichý's definition of intensions and extensions:

Definition 2.1 *Let α be a type and let $\{i_1, ..., i_k\}$ be a set of indexes with $i_j \in \{1, ..., n\}$. An intension is a $((((\alpha)\omega_{i_1})...)\omega_{i_k})$ object for modalities $\omega_{i_j} \in \Omega$ which partially associates α-objects to possible worlds. An intension is non-trivial if there are at least two possible worlds $w^1 = (w^1_1, ...w^1_n)$, $w^2 = (w^2_1, ..., w^2_n)$ such that the values of the intension are different at these worlds. All other first order objects that are not intensions are called extensions.*

In the following we will write an intension function as $f_{\omega_{i_1}...\omega_{i_k}}$. If the concrete modalities are not important, we write f_Ω. Consider a function $Employee_{\mu\tau\omega} = Employee/(((o\mu)\tau)\omega)$ which assigns a time line to each instance where at each point of time there exist multiple alternatives of being true or false. Please note that this is intensionally different to a function $Employee_{\omega\mu\tau} = Employee/(((o\omega)\mu)\tau$ which assigns alternatives to time lines where at each alternative we map instances to truth values. Simply spoken, the first function determines whether an instance may represent an employee at a certain point of time while the second one considers the possible sets of employees at certain points of time.
A classification of types that can be built using these rules can be found in [DM00]. The most important ones are:

- *Properties* of α-elements are denoted by functions $(o\alpha)_\Omega$ for a type α:

$$f(a \in \alpha)_\Omega = \begin{cases} true & , a \text{ has the property at this possible world} \\ false & , a \text{ has not the property at this possible world} \end{cases}$$

- *Propositions* are o_Ω functions that assign truth values to possible worlds.
- *Offices* (sometimes also called *individual concepts*) are ι_Ω functions that assign individuals to possible worlds.

Refered to our modalities of instances, time, and alternatives we can associate the following functions to modeling constructs known from the extensional point of view:

- Functions $o\mu\tau\omega$ can be seen as characteristic functions for classes, entity sets, or relationship sets, e.g. $Employee_{\mu\tau\omega}$:

w_1	t_1	t_2	t_3	t_4	...
m_1	T	T	T	T	
m_2	F	F	T	T	
...					

w_2	t_1	t_2	t_3	t_4	...
m_1	F	F	F	F	
m_2	T	T	F	T	
...					

...

- Functions $\iota_i\mu\tau\omega$ can be used to express attributes, e.g. $Salary_{\mu\tau\omega}$:

w_1	t_1	t_2	t_3	...
m_1	€4000	€4000	€4300	
m_2	€0	€0	€2000	
...				

w_2	t_1	t_2	t_3	...
m_1				
m_2	€2000	€2000	€0	
...				

...

- Functions $\omega\mu\tau\omega$ represent a relationship to another instance, e.g. $Boss_{\mu\tau\omega}$:

w_1	t_1	t_2	t_3	...
m_1	w_3	w_4	w_4	
m_2			w_4	
...				

w_2	t_1	t_2	t_3	...
m_1				
m_2	w_1	w_1		
...				

...

- Combining these patterns we can express set valued relationships $(o\omega)\mu\tau\omega$, e.g. $Inferiors_{\mu\tau\omega}$ (written in comprehensed form):

w_1	t_1	t_2	t_3	...
m_1	$\{w_5, w_6\}$			
m_2	$\{w_5, w_6, w_2\}$	$\{w_5, w_6, w_2\}$		
...				

w_2	t_1	t_2	t_3	...
m_1				
m_2	$\{\}$	$\{\}$		
...				

...

- Functions $(o\omega)\mu\tau$ express classes of instances, e.g. $Employees_{\mu}\tau$ (written in comprehensed form):

Employees	t_1	t_2	t_3	...
m_1	$\{w_1, w_4, w_5, w_6\}$	$\{w_1, w_4, w_5, w_6\}$	$\{w_1, w_4, w_5, w_6\}$	
m_2	$\{w_2, w_4, w_5, w_6\}$	$\{w_2, w_4, w_5, w_6\}$	$\{w_1, w_4, w_5, w_6\}$	
...				

Intension can be composed to form other intensions in a transparent way. [DM00] introduced four different types of *constructions*:

1. *Variables* of a type α
2. *Trivialization:* Let X be a construction or an object. ${}^0X$ constructs X without any change. Trivialization is needed for using objects in constructions and for mentioning (in difference to using) constructions.
3. *Composition* applies values to a function. Let X, $X_1, ..., X_n$ be constructions, $[XX_1, ..., X_n]$ is a construction called composition.
4. *Closure* creates functions. Let X be a construction, $[\lambda x_1, ...x_n X]$ is a construction called closure.

Semantics of constructions is given by valuations in the classical way: a valuation is a function which assigns to each variable of type α a α-object. Given a valuation v the variable x *v-constructs* the value $v(x)$. Constructions *v-construct* objects:

1. if x is a variable of type α, x v-constructs $v(x)$
2. if ${}^0X$ is a trivialization and X is a construction, then ${}^0X$ constructs (not v-constructs) X
3. Let $[XX_1...X_n]$ be a composition where X v-constructs $(\alpha\beta_1...\beta_n)$-objects and X_i v-constructs β_i-objects. If X v-constructs the function f and X_i v-construct the values b_i, then $[XX_1...X_n]$ v-construct the value of $f(b_1, ..., b_n)$. If f is not defined at $(b_1, ..., b_n)$ then this construction does not v-construct anything, it is v-improper. This is also the case when any X_i is v-improper.
4. Let $[\lambda x_1, ..., x_n X]$ be a closure construction where X v-constructs α-objects and (pairwise distinct) variables x_i v-construct β_i-objects. If $(b_1, ..., b_n)$ is a tuple of β_i objects, then v' is a valuation which is identical to v but assigns b_i to x_i. The value of the function f v-constructed by $[\lambda x_1, ..., x_n X]$ at $b_1, ..., b_n$ is the value v'-constructed by X. If X is v'-improper, f is undefined.

A construction is called a *concept* if it does not contain unbound variables.
Based on first-order types and constructions over first order types Tichý introduced the *ramified hierarchy* of types. Types of higher order allow construction that mention (not use) other constructions.

2.3.1 Imaginations as Descriptions of Valuations

Tichý's intensional constructions allow building the intension of expressions based on the intension of subexpressions. Given a construction that constructs the intension of an expression and a concrete valuation v it is possible to associate the intension behind an expression with concrete objects (truth values, individuals, modalities) in certain possible worlds.
But having such an intension about an expression or about the world is not enough for having an imagination about concrete objects. Certainly, it is possible to construct intensions down to the objectual base and primitive functions such that the valuation's properties become fully transparent. But this is not a proper representation because it blows up complexity even in simple cases. That's why transparency is usually only maintained up to a certain level of granularity. So we need a facility which provides at least a restricted transparency behind this level of granularity.
Use the construction for the expression "The king of France does not exist.":

$$\lambda w \lambda t[^0\neg[^0 E_{wt}[\lambda w \lambda t[^0 K^0_{wt} F]]]$$

The example was taken from [DM00] where a deeper analysis of this construction can be found. The construction defines the intension of the expression modulo a valuation for the functions used in the expression, namely $\neg$, E, K, and F. The relationship between the intension and the reflection of this intension in "real world" is not given by this construction because it is out of scope of intensional logic. A valuation which assigns e.g. the constant "true" function to $\neg$ is fully appropriate. For that reason a pure intensional approach to concepts is not capable of associating intensions and valuations that are considered "appropriate" according to experiences and the knowledge about the world.
If we take a look at the example above from a human's perspective, we can identify different types of intensions:

- Functions like $^0\neg$ are seen to be completely known. It is not considered that the valuation of this function will change according to our understanding of the world. We will reject any valuation that conflicts with our one.

- The valuation of functions like "king of France" may be partially known, e.g. we know that *Louis XIV* was a king from 1643 until 1715 and *Friedrich Wilhelm I* was never king of France. For other individuals this information may be missing. Additionally, if there are valuations conflicting with our knowledge, we may update our imagination of the world.

- There may be intensions with no or no structured extensional information. For example, one may have an intension of a "king" but may have no idea what is a king. Another one may have an intension of a "king", is able to recognize a king (which means that the intension function is computable), but is not able to characterize a "king".

- If we look at sets of intensions we have certain knowledge about implications between valuations of different concepts. For example, it is very likely (but not mandatory) that if the king of France exists, he resides in or near Paris.

- If an intension is described by a construction the valuation relies on the subconstructions. Decision whether a valuation for the constructed intension will be considered to be valid or not relies on decisions about the validity of valuations for subconstructions but may add new constraints. Constraints for the valuation of the constructed intension may be evaluated after or in parallel to the constraints of subconstructions. For example, one might associate the opinion about the existence of the king of France with the imagination of a king and of France. The assumption that the king of France exists may imply certain properties for the imagination of a king. On the other side, one may first think about what is a king and what is France and may subsequently think of the existence of the king of France. This kind of handling intensions potentially excludes valuations which are considered to be approriate in parallel evaluation but it is the usual human way to cope with complexity.

Based on these cases we can introduce the notion of imaginations

Definition 2.2 *Given a set F of intensions, partial valuations v_i for $f_i \in F$, a dedicated intension $f \in F$ with a signature $\alpha\beta_1...\beta_n$ and the corresponding partial valuation v, a imagination of f is a tuple (i_f, c_f) where*

- $i_f : 2^{\alpha\beta_1...\beta_n} \longrightarrow o$ *is called identification function, a partial characteristic function which determines whether v is an appropriate valuation for f or not.*

- $c_f : 2^{\alpha\beta_1...\beta_n} \longrightarrow 2^{\alpha\beta_1...\beta_n}$ *is called local completion function. It is a partial function defined for every v which is identified by i_f as an appropriate valuation and incorporates additional knowledge in v. We assume c_f to be idempotent: $c(c(v)) = c(v)$ and $i(v) = i(c(v))$.*

Consider the simple example intension K with signature $o\omega_\tau$ representing "The entity represented by $w \in \omega$ had been the king of France or will be the King of France at some point in time". We know that this property is independent of the current time. If we have a valuation v which assigns *true* ($false$) to some entity w at some point in time t and is undefined else we can complete v to v' such that $v'(w)$ is also *true* ($false$) for all points in time t'. If we have a valuation v which assigns *true* to some w at time t_1 and which assigned $false$ to w at time t_2 we know that this valuation cannot be an appropriate valuation for our intension.
The imagination of an intension describes meta knowledge about intensions. We will not assume imaginations to be sound and complete descriptions but to be mental reflections which may themselves be completed or revised.

2.3.2 Conceptual Contexts

We will now introduce *conceptual contexts* for associating intensions and imaginations. The main construct for conceptual contexts is the concept itself. In TIL concepts were introduced as constructions without free variables. For the reasons explained in section 2.3.1 we have to open this definition to allow free variables in a restricted way. These free variables will be bound in an imagination driven way.
In the following discussion the objectual base is given by:

- A set o of truth values containing at least the values $true$ and $false$.
- A non empty set of non empty classes of individuals ι_i^C with corresponding finite sets ι_i such that $(\forall j \in \iota_i)(j \in \iota_i^C)$.
- A set of non empty classes of modalities ω_i^C. with corresponding finite sets ω_i such that $(\forall w \in \omega_i)(w \in \omega_i^C)$.

For each class α in the objectual base we assume that an algebra with appropriate operations is given, e.g. a boolean algebra for truth values. Additionally, we assume a set R of relations between elements of classes α and between elements of different classes. At least the equality relations $=_\alpha \subseteq \alpha \times \alpha$ are element of R.

Definition 2.3 *The set F^0 of primitive functions is given in the following way:*

- *If f is an operation $\alpha \times \alpha \longrightarrow \alpha$ in the algebra for α, $f \in F^0$.*
- *If r is a relation in R then the characteristic function $r : \alpha_1 \times ... \times \alpha_n \longrightarrow o$ is a primitive function.*
- *Nothing else is a primitive function.*

Using the objectual base we can introduce intensional functions as TIL first order types, see section 2.3 as the base for defining the notion of concepts.

Definition 2.4 *A first level concept is a tuple $(sig, x, \{(x, im)\}, im)$ where*

- *sig is the definition of a first order type,*
- *x is a construction in terms of TIL; on the first level we only allow constructions consisting of a single variable,*
- *$\{(x, im)\}$ is a binding of variable x to an imagination im,*
- *im is an imagination, either*
  - *a tuple (i, c) consisting of an identification function i and a completion function c as defined below,*

– *an arbitrary valuation v assigning a sig-object to this concept; in this case the identification functions i of this concept's imagination assigns true to v and false to any other valuation; the completion function is the identity function for sig objects. The valuation can also be a primitive function $f \in F^0$.*

Because we only deal with simple variable constructions on the first level, the imagination of the concepts is equal to the imagination of the underlying function given in the binding.

We will denote the set of first level concepts by $\mathcal{C}^1$. Based on first level concepts we can build concepts of higher level using constructions as defined in TIL. We will allow "free" variables in constructions as long as the imagination of the concept can be built by binding these variables to concepts on lower levels.

Definition 2.5 *A first level construction template is a tuple $(sig, constr, \mathcal{X}, im)$ where*

- *constr is construction in terms of TIL using first level concepts as constants, and*
- *$\mathcal{X}$ is the set of free variables in constr, each $x \in \mathcal{X}$ is of type sig_x, and*
- *im is an imagination template for this construction as defined below.*

We will denote the set of first level construction templates with $\mathcal{K}^1$. The set of construction templates restricts the possibilities how concepts can be composed to derive concepts of higher level. The general idea is: a construction template of level l with free variables X (the "parameters" of this construction template) is instantiated by associating each $x \in X$ with a concept from level l or below. The construction *constr* of the construction template defines the intensional construction of the resulting concept while the rules of the imagination template *im* define how the imagination of this concept is built depending on the template's parameters.

Definition 2.6 *A concept on level l is a tuple $(sig, constr, binding, im)$ where*

- *sig is a first order type equal to sig of constr,*
- *constr is a construction template of level $l-1$,*
- *binding is a function which assigns each free variable of constr to a concept of level $l-1$,*
- *im is the imagination of the concept built from the imagination template of constr by applying the given binding.*

Definition 2.7 *A construction template of level l is a tuple $(sig, constr, \mathcal{X}, im)$ where*

- *sig is a first order type,*
- *constr is a TIL construction using constructions and concepts of level l as constants,*
- *$\mathcal{X}$ is the set of free variables in constr, and*
- *im is the imagination template.*

Considering sets of concepts and construction templates we can introduce the notion of a conceptual context: a conceptual context is a closed area with a finite number of predefined concepts and construction templates. Conceptual contexts should be open in two ways:

- Construction templates allow the deriviation of new concepts out of existing concepts, although possibilities are limited because the number of parameters for construction templates is finite.
- Valuations of concepts are only determined up to the concept's imagination. More specific concepts can be built by completing these imaginations.

Definition 2.8 *Let $\mathcal{C}$ be a set of concepts and $\mathcal{K}$ a set of construction templates such that $\mathcal{C}$ contains all concepts used in constructions of templates from $\mathcal{K}$ and $\mathcal{K}$ contains all constructions used to build concepts from $\mathcal{C}$. Then the tuple $\mathfrak{C} = (\mathcal{K}, \mathcal{C})$ is called a conceptual context.*

2.4 Adding Conceptual Information to Content Management Systems

When putting together these pieces we can enrich the presented semiotic triangle in the way depicted in figure 2.2.
Conceptual modeling of information systems is not a new idea, but todays systems usually "ignore " conceptual information when it comes to implementation. In the following chapters we want to discuss how conceptual information can be incorporated in content warehouses to be used in various tasks. Following the semiotic separation of concerns we can identify the following major components of such a system:

- The *content engine* is responsible for storing data. It corresponds to the structural part of the conceptual model. Following the element-driven approach the content engine's major task is to store information chunks independently from particular schemata.

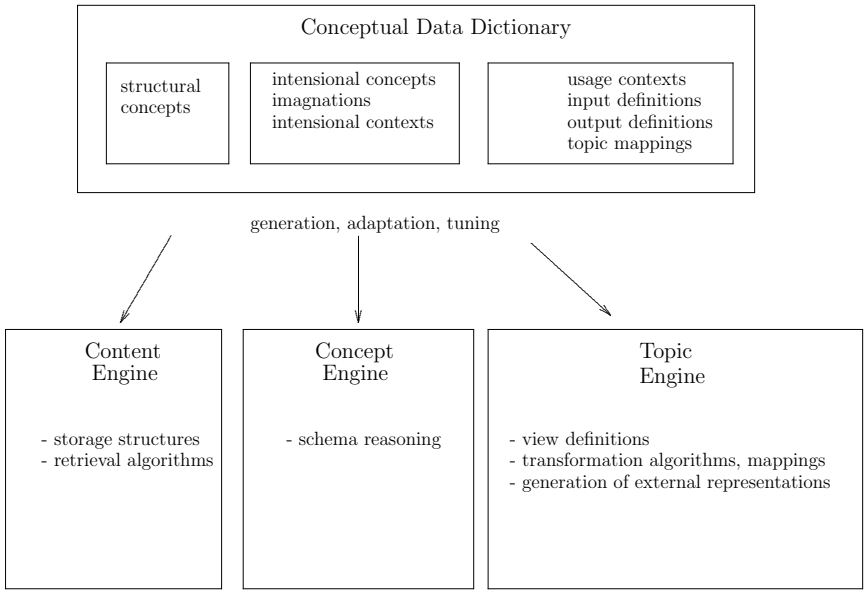

Figure 2.2: Components of a Content Warehouse According to the Separation of Concerns

- The *concept engine* realizes the access to data dependent on intensions and imaginations of content. This replaces the traditional notion of a schema: content fulfills a conceptual schema if it does not contradict to the concept's imagination within a chosen conceptual context.
- The *topic engine* controls the system's interface with its environment. It provides the definition of valid representations for content chunks and concepts to prevent the system of being flooded with conceptual heterogenous content. Topics as refinements of concepts map contextualized representations to system-known intensions and content structures. This allows an independent representation of internal concepts and external representations.

In the following chapters we want to discuss how this general architecture can be established for the case of evolving information systems with the need for schema cooperation and schema versioning. In chapter 3 the approach of aquiring and incorporating conceptual information in the case of schema integration is presented. This approach is manifested in chapter 4 for change management for schemata. The more concrete case of schema evolution is discussed in chapter 5. Chapter 6 shows the possible usage of the aquired conceptual information for deriving the parts of a "classical" content management system: storage structures and retrieval functionality like templates.

3 Schema Integration by Conceptual Abstraction

3.1 Motivation

Although schema integration in general is undecideable ([Con86]), it is usually achievable in practical schenarios. This observation can be based on a simple hypothesis: data structures under integration represent real world objects that are already related to each other in some manner[1]. If there are no relationships between the real world objects under consideration, there is usually no need for integrating the data structures derived during the modeling process.

If we assume this already integrated real world scenario, the modeling process as an indeterministic creative act of a modeling expert and the limitations of used database models and available database management systems cause heterogeneous data structures. Every step in data modeling introduces its own aspects.

In a first step, the "real world" is reduced to a "mini world" which covers all relevant aspects while the term "relevant" is defined locally by the modeling expert. Aspects not relevant in this scope may be relevant in other scopes or in a more general scope. This causes a first potential for integration needs.

The mini world is manifested in a formal conceptual model. Again, the modeling expert decides the mapping between elements of the mini world to available language constructs of the chosen modeling language, e.g. ER, HERM, or UML. Support of these constructs may be restricted due to the modeling tool or specified modeling styles, e.g. n-ary relationships in entitiy relationship models.

The logical layer may introduce new modeling artefacts, e.g. due to unnesting of attributes, normalization or denormalization in the relational data model. This is carried on on the physical layer where the designer defines data types, storage formats, or encodings.

Ad hoc schema mapping algorithms attempt to merge two (or more) schemata on a certain level of detail into a global one. Because the number of possible schemata representing the original modeling task increases from one layer to the next lower one according to the path rule it is much more difficult to integrate schemata on lower levels of abstraction than on higher ones. Artefacts introduced on a certain layer may cause conflicts that have to be solved during integration. For example, integrating

[1] "Jetzt wächst zusammen, was zusammengehört." — "It grows together what belongs together" (Willy Brandt)

schemata on the physical layer needs integration of storage oriented data types like integer values of different bit size or strings with different character encodings.
Beside the described enrichment "forgetful" design complicates schema integration on lower layers. It may be possible that artefacts are relaxed during development. For example, attribute domains on the conceptual or logical layer are replaced by data types that represent a superset of the intended domain, e.g. the set of all possible names for persons is mapped to the set of strings. Other examples are integrity constraints like foreign keys or more general ones that are not represented in the schema but in the application logic due to system or language restrictions, real (or felt) performance enhancements or cushiness. While enrichment makes integration more difficult, forgetful design may prevent it.
Formulating an integration aware design methodology is self-contradictory because all mentioned modeling aspects that cause problems during schema integration are mandatory for designing working systems in a pragmatical way. But to cope with these issues, a successful and sustainable schema integration should not be restricted to ad hoc schema mapping, but has to consider all decisions during the design process.
Given a set of schemata $\{\mathfrak{S}_1, ..., \mathfrak{S}_m\}$ schema integration aims to find a schema $\hat{\mathfrak{S}}$ such that there are mappings $f_1, ..., f_m$ that transform $\mathfrak{S}_i$ to constructs of $\hat{\mathfrak{S}}$ and $\hat{\mathfrak{S}}$ fulfills the criterias for a "good schema" (see [Tha00]: completeness, naturalness, minimality, system independence, and flexibility) as good as possible.
Schema integration by conceptual abstraction is not a mapping algorithm that contributes anything to ad hoc schema integration. It is a design methodology that aims in reducing the difficulties of forthcoming integration tasks. The methodology bases on a design process with ordered steps S_1 to S_n with $S_i \prec S_{i+1}$. On each step S_i there exists a schema representation $\mathcal{M}_i$ expressed in a language $\mathcal{L}_i$. An embracing schema $\mathfrak{S}$ is defined as the collection of all schema representations formulated during the the design process: $\mathfrak{S} = (\mathcal{M}_1, ..., , \mathcal{M}_n)$. The definition of schema integration can be directly adapted: The goal is a schema $\hat{\mathfrak{S}} = (\hat{\mathcal{M}}_1, ..., \hat{\mathcal{M}}_n)$ that subsumes the schemata $\mathfrak{S}_i$.
This definition assumes that every embracing schema $\mathfrak{S}_i$ is fully defined on all steps S_j. In practice, this is not very useful: especially the conceptual design is usually forgotten after the system becomes productive. Even if the conceptual model (ER or UML diagrams) remains for documentation purposes, the models are usually not represented in the system in a machine interpretable form. Additionally, it is not a very effective way to define all mappings between the local schemata $\mathcal{M}_j$ of $\mathfrak{S}_i$ and the local schemata $\hat{\mathcal{M}}_j$ of $\hat{\mathfrak{S}}$ independently.
Figure 3.1 describes the general tasks during integration:

1. At first, the layer S_i is identified where the integration should take place. This layer should be located as high as possible, but to enable any computational support it has to provide a formally defined language. Considering the steps of the design process above (mini world, conceptual model, logical model, physical

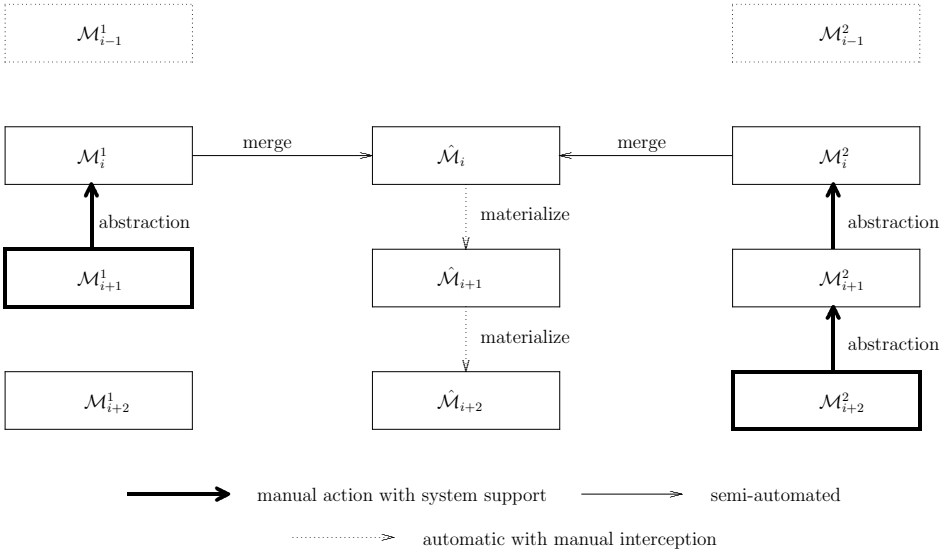

Figure 3.1: Schema Integration by Conceptual Abstraction

model), the conceptual model is the earliest step that provides a formalized specification.

2. Let $\{\mathfrak{S}_1, ..., \mathfrak{S}_n\}$ be the set of incomplete schemata that have to be integrated. For every schema $\mathfrak{S}_j$ the local schema $\mathcal{M}^j_i$ for the chosen integration layer is induced if it is not already present. This is usually a manual step with partial computational support.

3. The local models $\mathcal{M}^j_i$ are semi-automatically merged.

4. The integrated schema is materialized by deriving the models $\hat{\mathcal{M}}_k$ with $k > i$. Usually, automated translations generate appropriate schemata. For tuning reasons, this process can be supported by manual interception.

In the following, we manifest this general approach to the standard database engineering process as described above. We consider the conceptual layer as the layer of integration. Figure 3.2 shows the layering for this process. First, we will introduce a framework for defining the conceptual model. Later on, this framework will be used to accomplish conceptual integration.

3.2 The Meta-Concept Layer: Modeling High-Level Objects

On the meta-concept layer we localize the universe of discourse, the scope of the content warehouse. The managed content is seen as a set of information chunks.

Each chunk represents an entity in real world. We assume a global identification framework, either by system managed identificators or on the basis of key characteristics. In the following, we will call these information chunks *"objects"* and define a general representation frame. Let $\mathfrak{F}_1$, ..., $\mathfrak{F}_n$ be a set of (by now) abstract facets. The scope of the content warehouse is the set $\mathfrak{C} = \{\mathfrak{F}_1, ..., \mathfrak{F}_n\}$.

In other words, every object that will reside in this content warehouse can be modeled by expressing the facets $\mathfrak{F}_i$ in $\mathfrak{C}$. This frame is considered to be fixed. Changing the frame implies a system redesign. A typical frame for the observation example in our questionnaire application is depicted in figure 3.3. Each observation covers the facets "**who** was observing", "**what** was observed", "**where**, **when**, and **how** it was observed", and "which **result** was concluded from this observation".

Such frames are frequently used in data warehouse applications and correspond to star and snowflake schemata ([Tha05]).

To define the state of a content warehouse we consider the facets $\mathfrak{F}_i$ to $\mathfrak{F}_n$ to be sets of abstract facet elements. Therefore, each object can be represented by a tuple $\mathfrak{o} = (id, \mathfrak{f}_1, ..., \mathfrak{f}_n)$ where id denotes the global identificator for this particular object which makes it distinguishable from other objects. The object is associated with a facet element $\mathfrak{f}_i$ for every facet $\mathfrak{F}_i$. The state of the content warehouse is a set of objects: $\sigma(\mathfrak{C}) = \{(id, \mathfrak{f}_1, ..., \mathfrak{f}_n)\}$.

3.3 The Conceptual Layer: Concepts Under Consideration

The meta-concept layer defined the general scope of the content warehouse. Every object is expressed by an equally structured tuple. The conceptual layer refines this definition. A general discussion of an appropriate conceptual model can be found in chapter 2. Each facet of the meta-conceptual layer is replaced by a conceptual subschema, e.g. the conceptual definition of relationships with connected entities and relationships.

3.4 The Logical Layer: Defining Structure

The logical schema layer provides the connection between the conceptual layer defined above and the physical data representation. The data model on this layer depends on the available computational support. Usually, the relational data model is chosen, but object-oriented models and semistructured models might be appropriate, too. In the following discussion, we will restrict our attention to relational models.

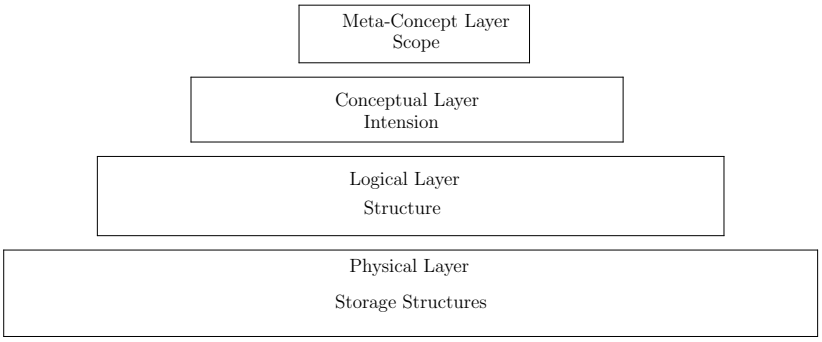

Figure 3.2: Abstraction Layers

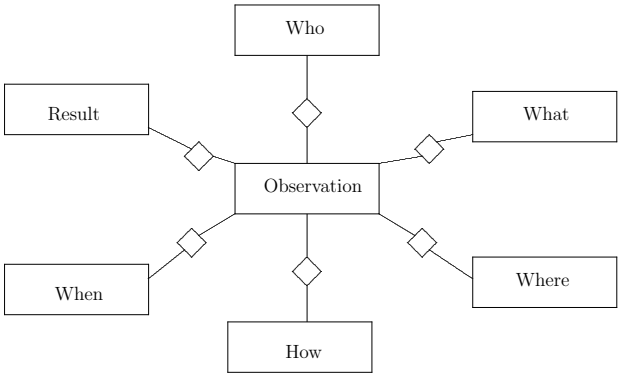

Figure 3.3: The Meta-Concept Layer for the Observation Questionnaire Application

3.5 Merging Conceptual Schemata

Merging two conceptual Models $\mathcal{M}^1, \mathcal{M}^2$ is mainly based on the notion of intensional containment. Intensional containment was studied in Kauppi's concept theory ([Kau67]) and subsequently e.g. in [Pal94], [Kan96], or [Nii04]. Given a reflexive and transitive relation $\mathfrak{C}_1 \sqsubseteq \mathfrak{C}_2$ it is said that the concept $\mathfrak{C}_2$ intensionally contains a concept $\mathfrak{C}_1$ or a concept $\mathfrak{C}_1$ is intensionally contained in concept $\mathfrak{C}_2$. Based on the relation of intensional containment, other relations between concepts $\mathfrak{C}_1, \mathfrak{C}_2$ are defined:

equivalence:	$\mathfrak{C}_1 \equiv \mathfrak{C}_2$	$\Leftrightarrow_{Df}$	$\mathfrak{C}_1 \sqsubseteq \mathfrak{C}_2 \wedge \mathfrak{C}_2 \sqsubseteq \mathfrak{C}_1$
strict containment:	$\mathfrak{C}_1 \sqsubset \mathfrak{C}_2$	$\Leftrightarrow_{Df}$	$\mathfrak{C}_1 \sqsubseteq \mathfrak{C}_2 \wedge \neg(\mathfrak{C}_1 \equiv \mathfrak{C}_2)$
immediate containment:	$\mathfrak{C}_1 \sqsubset_1 \mathfrak{C}_2$	$\Leftrightarrow_{Df}$	$\mathfrak{C}_1 \sqsubset \mathfrak{C}_2 \wedge \neg(\exists \mathfrak{X})(\mathfrak{C}_1 \sqsubset \mathfrak{X} \wedge \mathfrak{X} \sqsubset \mathfrak{C}_2)$
comparability:	$\mathfrak{C}_1 \text{ H } \mathfrak{C}_2$	$\Leftrightarrow_{Df}$	$(\exists \mathfrak{X})(\mathfrak{X} \sqsubseteq \mathfrak{C}_1 \wedge \mathfrak{X} \sqsubseteq \mathfrak{C}_2)$
incomparability:	$\mathfrak{C}_1 \text{ I } \mathfrak{C}_2$	$\Leftrightarrow_{Df}$	$\neg(\exists \mathfrak{X})(\mathfrak{X} \sqsubseteq \mathfrak{C}_1 \wedge \mathfrak{X} \sqsubseteq \mathfrak{C}_2)$
compatibility:	$\mathfrak{C}_1 \curlyvee \mathfrak{C}_2$	$\Leftrightarrow_{Df}$	$(\exists \mathfrak{X})(\mathfrak{C}_1 \sqsubseteq \mathfrak{X} \wedge \mathfrak{C}_2 \sqsubseteq \mathfrak{X})$
incompatibility:	$\mathfrak{C}_1 \curlywedge \mathfrak{C}_2$	$\Leftrightarrow_{Df}$	$\neg(\exists \mathfrak{X})(\mathfrak{C}_1 \sqsubseteq \mathfrak{X} \wedge \mathfrak{C}_2 \sqsubseteq \mathfrak{X})$

The general concept $\mathfrak{G}$ is a concept which is intensionally contained in every concept. According to Kauppi's theory of concepts, $\mathfrak{G}$ always exists, and therefore makes any two concepts $\mathfrak{C}_1$, $\mathfrak{C}_2$ comparable. A special concept $\mathfrak{S}_i$ is a concept which is not intensionally contained in any other concept except itself. There can be multiple special concepts. For every concept $\mathfrak{C}$ there is at least one special concept $\mathfrak{S}$ such hat $\mathfrak{C} \sqsubseteq \mathfrak{S}$.
Given two concepts $\mathfrak{C}_1$, $\mathfrak{C}_2$ the concept operations *intensional sum*, *intensional product*, *intensional negation*, *intensional reciprocal*, *intensional difference*, and *intensional quotient* can be defined, e.g.:

intensional sum:	$\mathfrak{A} \equiv \mathfrak{C}_1 \oplus \mathfrak{C}_2$	$=_{Df}$	$(\forall \mathfrak{X})(\mathfrak{A} \sqsubseteq \mathfrak{X} \Leftrightarrow \mathfrak{C}_1 \sqsubseteq \mathfrak{X} \wedge \mathfrak{C}_2 \sqsubseteq \mathfrak{X})$
intensional product:	$\mathfrak{P} \equiv \mathfrak{C}_1 \otimes \mathfrak{C}_2$	$=_{Df}$	$(\forall \mathfrak{X})(\mathfrak{X} \sqsubseteq \mathfrak{P} \Leftrightarrow \mathfrak{X} \sqsubseteq \mathfrak{C}_1 \wedge \mathfrak{X} \sqsubseteq \mathfrak{C}_2)$

We consider two schemata $\mathcal{M}^1$, $\mathcal{M}^2$ under intgegration. Merging $\mathcal{M}^1$ and $\mathcal{M}^2$ as depicted in figure 3.4 is a dialog styled process where the system perpetually suggests sets of concepts to be integrated together with the mapping of these concepts from the original schema to the integrated schema. The designer can accept or reject the integration proposals. This process runs until every concept from the original schemata is mapped to the integrated schema. On the other hand, the designer might choose concepts to be integrated while the system suggests an embedding of these concepts into the integrated schema.
If there are concepts left but integration is no longer possible from the point of view of the designer or the system, the designer derives local schemata $\tilde{\mathcal{M}}^i$ by

- refining the original schemata by adding or removing constraints. The system will assist by checking validity of already integrated parts. This step is especially necessary if the conceptual schema was induced from a logical schema or neglectfully established. Typical tasks are e.g. resolving domain conflicts

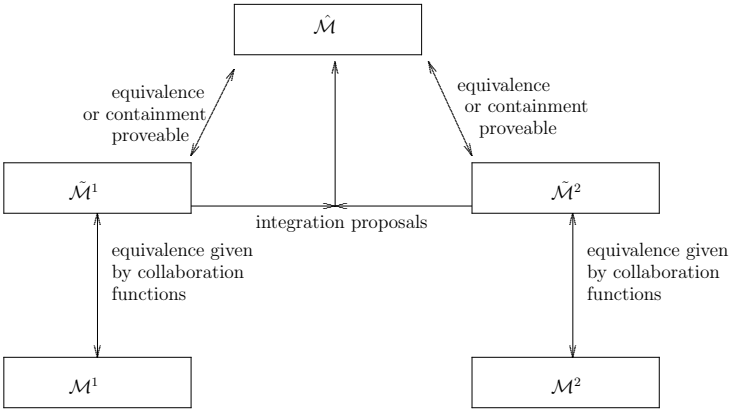

Figure 3.4: Merging Schemata

(break down all-embracing domains like `string`) or local synonym/homonym conflicts.

- introduce explicit dependencies between concepts of the schemata to be integrated, e.g. introducing new concepts to the integrated schema, declaring intensional containment between concepts, casting property domains, or deriving properties.

Semantical equivalence between local schemata $\mathcal{M}^i$ and $\tilde{\mathcal{M}}^i$ is maintained by conversion functions with possibly hidden semantics (see e.g. view collaboration functions in [FRT05]) and therefore, cannot be automatically proven.
Integration proposals can be based e.g. on the following relationships between concepts. We restrict our attention to the integration of two schemata $\mathcal{M}^1$ and $\mathcal{M}^2$, but the procedure can easily be extended to a set of schemata $\mathcal{M}^1, ..., \mathcal{M}^n$.

- *Identification of equivalent concepts:* If there exist concepts $\mathfrak{C}_1 \in \mathcal{M}^1$ and $\mathfrak{C}_2 \in \mathcal{M}^2$ with $\mathfrak{C}_1 \equiv \mathfrak{C}_2$, one of the concepts is added to the integrated schema $\hat{\mathcal{M}}$. Concepts $\mathfrak{C}_1$ and $\mathfrak{C}_2$ are directly mapped to the integrated concept $\hat{\mathfrak{C}}$ with respect to possible syntactical differences.

- *Identification of intensional containment:* If there are concepts $\mathfrak{C}_1 \in \mathcal{M}^1$ and $\mathfrak{C}_2 \in \mathcal{M}^2$ with $\mathfrak{C}_1 \sqsubset \mathfrak{C}_2$ (or vice versa), then a new concept $\hat{\mathfrak{C}}$ is introduced with all the characteristics of $\mathfrak{C}_2$. The parts of $\mathfrak{C}_2$ that are not present in $\mathfrak{C}_1$ are declared to be optional in compound. $\mathfrak{C}_1$ is mapped to $\mathfrak{C}$ with a missing optional part, $\mathfrak{C}_2$ is mapped directly. The condition that the optional part either can be missed completely or has to be present completely may be relaxed, though the resulting concept $\hat{\mathfrak{C}}$ becomes weaker than the original concepts $\mathfrak{C}_1$, $\mathfrak{C}_2$.

- *Identification of comparable concepts:* $\mathfrak{C}_1$ и $\mathfrak{C}_2$: the integrated concept $\hat{\mathfrak{C}}$ will contain the kernel concept $\mathfrak{X}$ common to $\mathfrak{C}_1$ and $\mathfrak{C}_2$ as mandatory characterisitics as well as the characteristics specific to $\mathfrak{C}_1$ and $\mathfrak{C}_2$ as mutual optional

characteristics. Mutual optionality (exactly one part is missing) may be relaxed to produce a weaker integrated concept $\hat{\mathfrak{C}}$.

Integration of two comparable concepts can be extended: if $\mathfrak{C}_{1_1} \sqcap ... \sqcap \mathfrak{C}_{1_n} \equiv \mathfrak{C}_2$, the characteristics of concept $\mathfrak{C}_2$ are used as mandatory kernel characteristics which may be extended by the (now mutually optional) characteristics of $\mathfrak{C}_{1_1}$ to $\mathfrak{C}_{1_n}$.

Because the general concept $\mathfrak{G}$ is intensionally contained in every concept, integration by comparability is always possible, but in this case the integrated concept will degenerate to a concept which contains the mutually optional characteristics of $\mathfrak{C}_1$ and $\mathfrak{C}_2$ without any intersection.

4 Change Management And Harmonization of Conceptually Enriched Schemata

After applying the procedures presented in chapter 3 we assume for every schema considered to be under integration there is an embracing schema $\mathfrak{S}^j$ available that is fully specified at least on all layers below (and including) the chosen layer of integration. After these schemata were already subject to integration and therefore subject to changing requirements, it is obvious that changes will also occur in the future.
We assume the schemata under integration to be independent from each other. Although schemata were integrated into a global one and / or a collaboration network with mappings was established, each schema may be subject to a local change management. If local changes occur the collaboration framework will collapse with another need for additional integration efforts.
We can use our information about relationships between local schemata to track local changes and to propagate these changes through the collaboration network to keep the global integrity. In this chapter we will discuss a framework of expressing schema change operations as production rules of the specification language. We will discuss to specify mappings between the production rules of different languages on different levels of granularity.

4.1 Compatibility Maintenance Between Schemata

4.1.1 Faithful Embracing Schemata and Faithful Integrated Systems

The situation we discuss here is depicted in figure 4.1 with the relevant layers for the case of the mentioned database engineering process. We consider the binary case of two collaborating schemata $\mathfrak{S}_1$, $\mathfrak{S}_2$ with the conceptual layer as the layer of integration.
Changes occur in an embracing schema $\mathfrak{S}_i$ on a certain level of abstraction ($\mathcal{M}^i_j$). These changes have to be propagated

1. within the embracing schema $\mathfrak{S}_i$ to other levels of abstraction,
2. between embracing schemata, in our case between $\mathfrak{S}_1$ and $\mathfrak{S}_2$.

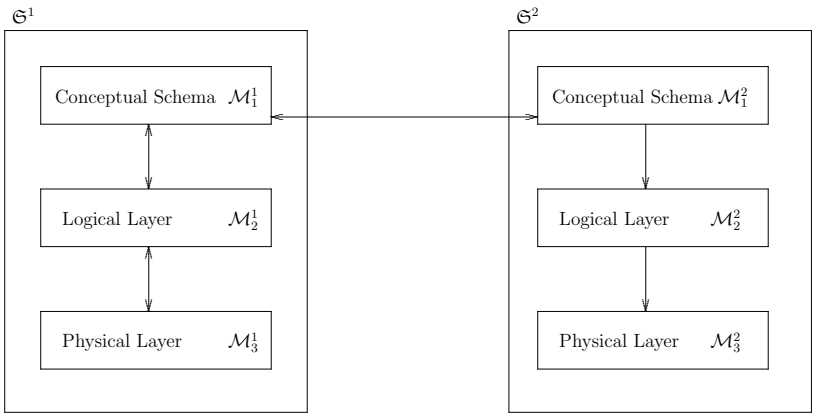

Figure 4.1: Change Propagation

For the second case we assume a hub oriented structure: changes between embracing schemata are only delivered via the layer that was chosen for integration. For example, if changes were made in schema abstraction $\mathcal{M}_2^1$ the changes are first propagated to $\mathcal{M}_1^1$ and $\mathcal{M}_3^1$. Afterwards, the changes implied on $\mathcal{M}_1^1$ are propagated to $\mathfrak{S}^2$ via the connection between $\mathcal{M}_1^1$ and $\mathcal{M}_1^2$.
We can classify the relationships between schemata in the following way:

- *Change Sensitivity:* Schema changes may not be allowed in every possible embracing schema $\mathfrak{S}_i$ or every schema abstraction $\mathcal{M}_j^i$. For example, $\mathfrak{S}_2$ defines an environment where changes should be allowed on the conceptual layer to schema abstraction $\mathcal{M}_1^2$, but any change that is not implied by a change operation on $\mathcal{M}_1^2$ is strictly forbidden. The conceptual layer *dominates* the subsequent layers. Analogously, dominant embracing schemata can be introduced.

  On the other hand, in $\mathfrak{S}_1$ all layers are considered to be equally treated: changes can be made on every level of abstraction while all changes are propagated to the other layers. Schema abstractions $\mathcal{M}_j^1$ are considered to be *equivalent in terms of change sensitivity* if they dominate each other.

- *Scope:* Schemata may be of different scope which means that changes in one schema do not affect or only partially affect the other schema because the intuition behind the changed artifacts is not present in the target schema. On the other hand, the target schema may be of richer scope than the source of change. In this case change propagation does not fully determine the change in the target schema in a possible world sense.

- The usual relations between languages in terms of expressiveness apply.

We extend our notion $\mathfrak{S} = (\mathcal{M}_1, ..., \mathcal{M}_n)$ of embracing schemata by a mapping function $m_{\mathfrak{S}}$ for change operations between schema abstractions. Analogously, we intro-

duce the notion of an integrated system $\mathfrak{I} = (\mathfrak{S}_1, ..., \mathfrak{S}, m_{\mathfrak{I}})$ of embracing schemata $\mathfrak{S}_i$. We call an embracing schema $\mathfrak{S}$ *faithful*, if

1. The languages on all layers are equivalent in terms of expressiveness.
2. The set of dominant layers in terms of change sensitivity contains the layer of integration.
3. For each dominant layer in terms of change sensitivity the mapping function m is defined for mappings to all other schema abstractions.

We call an embracing schema $\mathfrak{S}$ *fully faithful*, if

1. $\mathfrak{S}$ is faithful,
2. all schema abstractions are equivalent in terms of change sensitivity,
3. m is deterministic

Fully faithful embracing schemata allow a free choice of the specification language modulo a global configuration for deterministic scope handling.
We call an embracing schema $\mathfrak{S}$ *fully operational faithful*, if

1. $\mathfrak{S}$ is fully faithful,
2. For any two change operations c on $\mathcal{M}_i$ and $\tilde{c}$ on $\mathcal{M}_j$ it holds that

$$m(\mathcal{M}_i, \mathcal{M}_j)(c) = \tilde{c} \quad \Leftrightarrow \quad m(\mathcal{M}_j, \mathcal{M}_i)(\tilde{c}) = c$$

Similiar definitions can be found for an integrated system $\mathfrak{I}$. In the following discussion we will restrict our attention to faithful embracing schemata and faithful integrated systems.

4.1.2 Change Propagation Based on Language Production Rules

Specification of Schema Abstractions

The notion of a faithful integrated system[1] determines the existance of necessary mapping facilities between schemata. To use these facilities for change management we have to express the anatomy behind the mapping function.
At each schema abstraction in each embracing schema a certain language $\mathcal{L}_j^i$ is spoken. Because modeling languages are usually visual languages we consider $\mathcal{L}_j^i$ to be a graph language with a graph grammar $G_j^i = (\Sigma, P, S)$. Σ contains sets of (terminal and non-terminal) node labels and edge labels. We further distinguish labels of both categories in the following way:

[1] In the following when we are going to speak about integrated systems we automatically include embracing schema in the discussion. If we use the term schema abstraction in a context of an integrated system, we always refer to the layer that was chosen for integration.

- *language level:* non-terminal labels representing types of language constructs like "entity type" or "relationship type" in HERM. Additionally, we allow also terminal language level labels.
- *schema level:* non-terminal and terminal labels representing schema specific elements, e.g. the entity type "Person" in a certain HERM schema.
- *change level:* non-terminal labels denoting positions where schema changes may occur.

Further on, the grammar facilities a start graph S and a set of production rules $P = \{r_i \ : \ \langle L \supseteq K \subseteq R \rangle\}$. Production rules can be classified in the following way:

- *Schema building rules* describe creation of a schema our of the start graph. Schema building rules are formulated over the labels on language level.
- *Schema generating rules* transform non-terminals from the language level to non-terminal labels from the schema level.
- *Schema instantiating rules* transform schema level non-terminal labels deterministically to schema level terminal labels and to non-terminal labels from the change level. We assume a one-to-one-to-one mapping between schema level non-terminal labels, schema level terminal labels, and change level non-terminal labels. For each triple $(X_k, x_k, \tilde{X}_k)$ of a schema level non-terminal label X_k, a schema level terminal label x_k, and a change level non-terminal label $\tilde{X}_k$ the following schema instantiating rules exist:

  X_k is replaced by x_k saturation rule
  X_k is replaced by $\tilde{X}_k$ change position marking rule

  Other schema instantiating rules do not exist.

- *Schema changing rules* describe valid change operations on a schema based on $\tilde{X}_k$ markers. There are two different approaches to define the set of schema changing rules

  1. The schema building rules can be also used as schema changing rules. In this case schema changing usually contain "lifting rules" that lift change position markers to non-terminal labels of "higher abstraction". Afterwards, usual schema building rules, schema generating rules, and schema instantiating rules are applied. So a change operation on a schema is considered to be a sequence of (in that order)
     a) lifting rule applications,
     b) schema building rule applications,
     c) schema generating rule applications,
     d) schema instantiating rule applications.

2. A fixed (and usually restricted) set of schema changing rules is defined. This set is adapted to needs in concrete application scenarios. These (usually unimportant) restrictions allow better utilization of change properties as well as clearer and better user interaction facilities. We will use this strategy for our questionnaire application.

A deeper discussion of graph grammars can be found in [EKMR99]. Application of graph grammars to ER schemata is available in [Tha01].

A *schema abstraction* $\mathcal{M}_j^i$ is a graph derived from S ($S \longrightarrow_P^* \mathcal{M}_j^i$) that only contains terminal labels and change level non-terminal labels. A saturated schema abstraction $\mathcal{M}_j^i$ is a schema abstraction only consisting of terminal labels.
Non-terminal labels in schema abstractions work as markers for positions were schema changing operations may occur. The set of schema changing production rules has to contain at leat the saturation rule that replaces $\tilde{X}_k$ by x_k for every change level non-terminal label $\tilde{X}_k$.
Grammars for handling change operations are usually context sensitive ones. To handle contexts in a proper way we guard our graph grammar by a context marking language $\mathcal{L}_j^{i,C}$. We extend the production rules to contextualized production rules. Production rules can be positively contextualized or negatively contextualized. A positively contextualized production rule is a production rule $r \quad : \quad \langle E, L \supseteq K \subseteq R \rangle$ such that transformation takes place within a subgraph which can be derived by $\mathcal{L}_j^{i,C}$ starting at E. A negatively contextualized production rule is a production rule $r \quad : \quad \langle \neg E, L \supseteq K \subseteq R \rangle$ such that transformation does not take place within a subgraph that can be derived by $\mathcal{L}_j^{i,C}$ starting at E.

Associating Production Rules

In our integrated system we now consider two schema abstractions $\mathcal{M}_j^i$, $\mathcal{M}_j^k$ where a change operation in $\mathcal{M}_j^i$ occurs. Both schema abstractions need to be harmonized. Each change operation in $\mathcal{M}_j^i$ corresponds to a sequence of applications of production rules of $\mathcal{L}_j^i$. To keep both schema abstractions harmonized, we need some change operation for $\mathcal{M}_j^k$ that reflects the changes in $\mathcal{M}_j^i$. This change operation will naturally correspond to a sequence of production rule applications. So we can define an equivalence relation between sequences R_1 of production rules from $\mathcal{L}_j^i$ and sequences R_2 of production rules from $\mathcal{L}_j^k$:

$$R_1 \equiv_{L_j^i, L_j^k} R_2 \iff R_1, R_2 \text{ reflect the same change operation in } \mathcal{M}_j^i, \mathcal{M}_j^k$$

When schema abstraction $\mathcal{M}_j^i$ dominates schema abstraction $\mathcal{M}_j^k$ and not vice versa the equivalence relation is relaxed to an implication:

$$R_1 \Rightarrow_{L_j^i, L_j^k} R_2 \iff R_1 \text{ is reflected in } \mathcal{M}_j^k \text{ by } R_2$$

In the following we will treat equivalence as two correlated implications of the form $R_1 \Rightarrow_{L_j^i, L_j^k} R_2$, $R_2 \Rightarrow_{L_j^i, L_j^k} R_1$.
The implication between sequences of production rules is expressed by a change propagation language[2] $\mathcal{L}_{i,k}^P = (T_{i,k}^P, N_{i,k}^P, P_{i,k}^P)$. A starting symbol is not needed, the production rule sequence triggered at $\mathcal{L}_j^i$ is considered as the starting sequence for the actual derivation. The non-terminal symbols of $\mathcal{L}_{i,k}^P$ contain the production rules of $\mathcal{L}_j^i$, the terminal symbols of $\mathcal{L}_{i,k}^P$ are exactly the production rules of $\mathcal{L}_j^k$.
Based on $\mathcal{L}_{i,k}^P$ we can redefine our implication relation between R_1 and R_2:

$R_1 \Rightarrow_{i,k} R_2 \iff$ Starting from R_1 the sequence of production rules $R_2 \in \mathcal{L}_{i,k}^P$ can be derived.

Looking at the derivation we can identify the following cases:

1. Starting with R_1 the sequence of production rules $R_2 \in \mathcal{L}_{i,k}^P$ is derived. So we know that the change operation R_1 is reflected in $\mathcal{L}_j^k$ by R_2. Now we can apply the production rules on $\mathcal{M}_j^k$:
   a) Applying the sequence of production rules R_2 on $\mathcal{M}_j^k$ results in a valid and (possibly, but not necessary) new schema abstraction $\tilde{\mathcal{M}}_j^k$. In this case the propagation is considered to be successful.
   b) Applying the sequence of production rules R_2 on $\mathcal{M}_j^k$ did not result in a valid schema abstraction, e.g. the process of derivation stopped with non-terminal symbols not located at the change level or the sequence of production rules could not be fully applied. The propagation of R_1 to $\mathcal{M}_j^k$ failed and is therefore forbidden within the integrated system.
2. The sequence of production rules R_1 is derived to the empty sequence R_2. This case is already included in the case discussed immediately before, but it is mentioned in relation to modeling scopes as discussed in section 4.3.1.
3. It is not possible to derive any sequence of production rules out of R_1. The change operation is considered to be failed and therefore forbidden within the integrated system.

An integrated system is called *harmonizable* if for every change operation that is possible in any dominating schema abstraction change operations can be found for every schema abstraction by applying the production rules of the defined change propagation languages.
An integrated system is called *strictly harmonizable* if it is harmonizable and in any situation the following condition holds for any change operation *op*:

op can be successfully applied $\iff$ all change operations derived by propagation of *op* can be successfully applied.

[2] We shorten the subscript from L_j^i, L_j^k to i, k to improve readability.

4.2 Deriving Associations Between Schema Elements

Application of production rules is embedded in a concrete schema abstraction graph. Based on association decisions and change propagations before we already know which component in the source schema abstraction is associated with which set of components at the target side. Applying a production rule on source side has to take place at the right position in the target schema abstraction to have the desired effect.
The mechanism of labeled glueing already introduced in graph grammars will support this process. Each production rule on source side facilitates a labeled graph structure to be replaced, a labeled glueing graph, and a labeled graph that will replace the first one. The same situation is at the target side. Additionally, we need a labeling connecting the elements on the level of propagation rules. This labeling actually determines which change propagations possible and therefore may be allowed in a harmonized integrated system.
Consider a set of labels $l_1^s, ..., l_n^s$ in the starting graph at the source side. Each labeled element e_i^s corresponds to labeled elements $e_{i,1}^s, ..., e_{i,m}^s$ in the produced graph at the source side. The labeling does not need to be a total mapping. The target side is labeled in a similiar manner with labels $l_1^t, ..., l_k^t$ and $l_{j,1}^t, ..., l_{j,l}^t$ respectively. The connection between elements from the source side and the target side is also labeled. The labels at the left side of the target rule have to consider also one-to-many relationships from the source to the target schema.
Please note that the mapping from labels to elements does not need to be injective. The situation is as follows:

1. The production rule at source side is considered to be applied. The rule matches a certain subgraph within the source schema abstraction. Labels l_i^s in the rule are matched to elements in the schema abstraction graph.

2. Each element labeled by l_i^s is connected to a set of elements $e_{i,1}, ..., e_{i,h}$ at the target side. If there is a one-to-many relationship we also know (or have to know) the labeling of the connections. In the case of one-to-one mappings this labeling is actually not of interest. Based on these two mappings we can match the left side of the target rule to the current schema abstraction graph at target side. If this embedding is not possible in a deterministic way, the propagation rule is considered to be failed.

3. Rules on source side as well as on target side are applied to obtain the new schema abstraction versions on both side. In a last step labeled connections between the new elements have to be established. We have a set of labeled elements on the source side which corresponds to a set of labeled elements on the target side. In the simplest case, all elements on one side correspond to all elements on the other side. For better granularity, additional labels are introduced to build equivalence classes within each set of elements. Elements

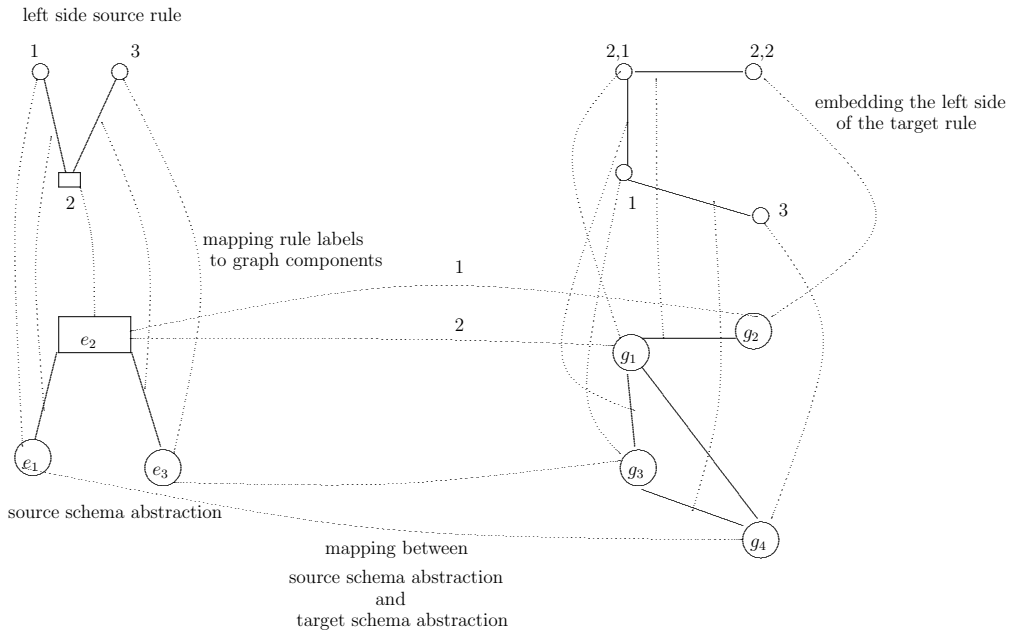

Figure 4.2: Applying Propagation Rules: Labeling

in an equivalence class are connected to all elements in the equivalence class at the other side with the corresponding label.

Figure 4.2 shows an example for labeling. During sequences of production rule applications labels are shared between the rules according to the derivation tree, e.g. associating non-terminals of "higher abstraction" with concreate graph components.

4.2.1 Handling Non-Determinism

It may be possible that change operations taking place in a source schema abstraction may be represented by different but equivalent change operations in a target schema abstraction. For example, consider an entity type in a HERM schema and an alternative XML Schema representation. When adding an atomar attribute to the entity type it is possible to nest this attribute as an element child into the element corresponding to the entity type. Alternatively, the attribute could be mapped to an attribute of the element which corresponds to the entity type.
More generally, we consider a sequence of production rule applications R_1 in the source schema abstraction and a set of sequences of production rule applications $\mathcal{R}_2$ such that each sequence from $\mathcal{R}_2$ reflects R_1 in terms of section 4.1.2.
Each sequence from $\mathcal{R}_2$ is a possible candidate for propagation. Based on secondary requirements there may be a preference of certain ways of change propagation. Considering the example, there may be a general guideline that XML attributes are prefered to XML elements whenever possible. So we add a preference relation between sequences of production rule application:

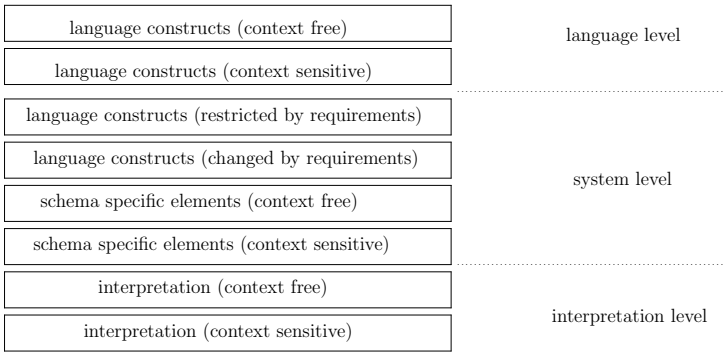

Figure 4.3: Granularity Levels for Change Propagation Rules

$$R_1 \prec_p R_2 \iff R_2 \text{ is preferred to } R_1$$

Definition of preference relation $\prec_p$ has to occur during system configuration to fit current needs from the system's requirement specification. If alternatives exist where no distinction is possible based on the preference relation one alternative has to be semi-automatically chosen.

4.3 Rule Granularity

The rules of the change propagation language apply on different levels of granularity, as figure 4.3 depicts:

- *language level:* Rules may be expressible for certain categories of the modeling languages, e.g. all entity types may be mapped to element type declarations. These rules can be context free or context sensitive like: "named element types are mapped to entity types, nested element types are mapped to tuple attributes." Rules on language level are applicable in any situation where a mapping between the two incorporated languages occurs.
- *system level:* Rules may depend on secondary requirements present in the system's scope and organisational or technical environment. This may happen in the following ways:
  - Rules from the language level may considered to be inappropriate or preference of rules may differ from the general assumption. For example, there may be the rule that base attributes of HERM entity types are mapped to attributes of the corresponding element type. Due to the requirements specification these attributes have to be mapped to nested element types. So rules on language level may be removed or may be overridden by new rules on system level.

  - There may be new dependencies possible in a concrete configuration. For example, the concrete entity type `Person` with attributes `name`, `birthday`, `address(street,zipcode,city,country)` may be represented by an element type `Person` with attributes `name` and `birthday` as well as an element type `Address` with attributes `street,zipcode,city,country` that is referenced by the `Person` element type.
    Similiar to language level rules system level rules may be context free or context sensitive.

  While rules on language level depend only on the modeling languages itself, system level rules have to be configured for every integrated system during system setup.

- *interpretation level:* Language level and system level rules require a certain kind of schema regularity for example derived by a schema driven development. But these rules are not able to cope with irregularities on instance level. For example, you have XML elements defining Persons:

```
<person name='John Smith' />
<person name='Mary Green' />
<person firstname='Joe' lastname='Black' />
```

  Irregularities on interpretation level can be of different kind. You might consider to develop two different schemata for the example: one person schema supplying a `name` attribute and one person schema supplying two attributes `firstname`, `lastname`. In this case language level and system level rules apply. But in the case of element driven development with minor or even without schema control the number of possible schemata grows exponentially. Although big parts of data can be handled by a quite small number of simple schema variants, there will be outliers with the need for individual schemas or increased schema variant size.

  It is more efficient to handle outliers on the level of interpretation itself: every outlier supplying irregularities that does not fit into the schema have to facilitate mapping rules on their own. The grammars introduced above have to be extended to allow derive interpretations, e.g. based on the general frame presented in section 3.3. Although we can use the general notion of a propagation language on interpretation level, rules will usually be of simple structure.

Rule granularity has to be taken into account while applying production rules during harmonization. We define a partial ordering relation $r_1 \prec_l r_2$ on rules of the change propagation language:

$$r_1 \prec_l r_2 \quad \Longleftrightarrow \quad r_1 \text{ is more specific than } r_2$$

As a necessary condition, "$\prec_l$" is strictly ordering between levels: interpretation level rules $\prec_l$ system level rules $\prec_l$ language level rules. Optionally, "$\prec_l$" may be defined for rules at the same level:

$$r_1 \prec_l r_2 \iff (\forall G)((G \rightarrow_{r_2} \tilde{G}) \Rightarrow (G \rightarrow_{r_1} \hat{G}))$$

If there are rule alternatives $r_1, ..., r_2$ at a certain step of derivation, alteratives are applied in order with respect to $\prec$ until a sucessful derivation could be established. If there is no ordering between rule aternatives r_1, r_2 derivations using r_1 and r_2 are seen as alternatives which may be distinguished by the preference relation defined in 4.2.1.

4.3.1 Handling Modeling Scopes

Looking at different schemata usually implies to use different restricted points of view. Additional to parameters based on language constructs the schema's modeling scope also determines available artifacts. In section 3.2 the notion of meta concepts was introduced at least for star and snowflake schemata facilitated by data warehouse systems. An example was given concerning a frame "who, what, when, where, how, result". Although it extends the scenarios discussed before we should take a look at harmonizing schemata residing within different scopes.
Consider the following case: We have a HERM schema $\mathfrak{S}_1$ built from the meta concept frame $\mathfrak{C}_1$ as introduced above and another XML schema $\mathfrak{S}_2$ that was built from a frame $\mathfrak{C}_2$ equally to "who, what, when, where, result". We restrict our attention to the case $\mathfrak{C}_2 \sqsubset \mathfrak{C}_1$ while the other cases apply in a similiar manner.
At a certain point of time schema $\mathfrak{S}_1$ is modified in the "how" facet by adding a new entity type `Hunt` together with a couple of attributes. According to our harmonization strategy we also have to add an element type corresponding to `Hunt` to the XML schema. $\mathfrak{S}_2$ now facilitates a `Hunt` element which expresses a method of raising observation data. Methods of raising observation data is out of scope $\mathfrak{C}_2$ and therefore, `Hunt` in inappropriate within schema $\mathfrak{S}_2$.
On the other side, if schema $\mathfrak{S}_2$ is modified in the "what" facet by adding an element type `WildBoar` the propagation takes place as usual, but associations of the new element type to the part of the schema representing the "how" facet will be missing in the propagation.
Considering scope spanning integration needs an extension of notion of propagation as defined above. Each propagated change operation from a schema $\mathfrak{S}_1$ to a schema $\mathfrak{S}_2$ is seen to be partially defined with a determined part considering the facets in the intersection of $\mathfrak{C}_1$ and $\mathfrak{C}_2$ and possible worlds according to the facets from $\mathfrak{C} - (\mathfrak{C}_1 \sqcap \mathfrak{C}_2)$. The possible worlds have to be manifested afterwards to complete the change operation.

4.4 Integration Scenarios

Putting together the pieces presented in the last chapters we can categorize supported schema integration in different but correlated fashions. We will discuss

relationships between schema level and instance level when applying integration facilities.
In the following we assume that at least partially a schema driven development strategy was used at is was described in chapter 3. A partially schema driven development strategy does not require full dominance of the schema over the instances, but we require that each instance e is associated with at least one distinguished schema $\mathfrak{S}$ such that $e \models \mathfrak{S}$ in the local manner. In different schemas different versions of this instance will exist. This condition may also be relaxed to a full element driven development strategy, but instance propagation in the subsequent discussion will not work in this configuration.
On schema level, three different notions of "integrated" schemata can be identified when using the facilities from the last chapters:

- *Associated schema abstractions* were subject to schema integration by conceptual abstraction. View collaboration functions were defined to derive schemata such that reasoning about intensional containment between schema components is supported. Instance representations can be calculated based on the intensional containment relationship on the schema layer. So instances residing in one schema can be (at least partially) propagated to the associated schema.

- *Harmonized schemata* are coupled schemata where coupling is used to propagate changes on schema level between schemata. This is especially useful for schema abstractions representing the same universe of discourse in different manners like conceptual schemata, logical schemata, and physical schemata used during the development process. If schemata to be harmonized are already present before harmonization starts, the schemata need some kind of association before, e.g. by schema integration by conceptual abstraction.

  The harmonization process as described before supplies information about how schema elements as well as instances are associated based on the correlated creation process. Harmonized schemata are associated schemata, too.

  Harmonized schemata are the only category where change operations can occur on schema level without further manual interception. Changing associated schemata in general usually needs adaptation of view collaboration functions afterwards.

- *Derived schemata* $\mathfrak{S}_d$ are built from other schemata $\mathfrak{S}_i$. Changes of schemata will not result in changing the correlated schemata, but in changes of propagated instances on the instance level.

Looking from the point of view of instances we can distinguish the followings kinds of integration:

- *View based retrieval:* Instances are kept independently from their representation in different schemata. The notion of "schema" is used like the notion of

"views" in classical database design except that we do not enfore an explicit all-embracing schema below the "views".

- *Object propagation:* View based retrieval in the simple form checks for satisfaction of instances in different schemata: $e \models \mathfrak{S}_i$ where satisfaction is defined in the local manner of $\mathfrak{S}_i$. If correlations between schemata become non-trivial the notion of satisfaction needs to be changed to $p_{\mathfrak{S}_i}(e) \models \mathfrak{S}_i$ for some propagation (or "view") function that is obtained from schema dependencies.

5 Schema Evolution

In the following section we will discuss an application scenario where advantages of schema integration by conceptual abstraction are revealed: schema evolution in content warehouses. A major bottleneck of schema integration by conceptual abstraction is the potentially great amount of manual interception during the integration process: conceptual schemata have to be extracted from the logical or physical schemata or even from the data and during the merging process the conceptual schemata have to be syntactically adapted to speak about the universe of discourse in comparable languages. The approach presented above considers integration of arbitrary schemata. If schemata are related to each other, e.g. they originate from the same state of modeling or represent observable variations from the same universe of discourse, knowledge about these relationships can be used to automate the manual parts of conceptual schema integration.
Talking about schema evolution we have to distinguish two different kinds of evolution, which may occur in combination:

1. The objects under consideration are established at certain points of time and remain in a fixed structure and/or state. The schema which is used to access the objects changes over time. This scenario can be compared to the classical notion of views over databases except that the system keeps track about relationships between views. Schemata may be ordered chronologically: objects created at a certain point of time t are only accessed using schemata defined at points of time $t_i \geq t$. We will assume the general case: at any point of time multiple schemata can exist, objects are created using a schema $\mathfrak{S}$ at a certain point of time and remain accessible by any other schema $\tilde{\mathfrak{S}}$ defined in the system.

2. The objects under consideration change their structure over the time. The (concurrent) schemata used to access the objects remain stable.

We will mainly consider the first case. Objects are created at a certain point of time t_c using a schema $\mathfrak{S}$ that is available at this time. They remain structurally unchanged until they are deleted at time t_d. The schema $\mathfrak{S}$ that was used to create a certain object o will be called *home schema* of object o. Because deletion of objects is not in the scope of our example application, we will not discuss this here, but it is naturally included. Like in any other version control system, the history of schemata will be kept forever. Deletion of schemata will never occur, unnecessary schemata may be hidden for pragmatical reasons.

facet	concepts
who	hunter, biologist
what	fox, crow, rabbit, badger
where	reference district, hunting district
how	count, hunt
when	timestamp, duration
result	statistics

Table 5.1: Example Concepts for the Questionnaire Application

Conceptual changes of objects are subject to *object migration* between schemata $\mathfrak{S}_1$, $\mathfrak{S}_2$. Migration of objects between schemata is not tracked by the system in terms of revision control and equals to a delete operation of object o in schema $\mathfrak{S}_1$ and a create operation for o using schema $\mathfrak{S}_2$. Any two objects o_1, o_2 existing in parallel are considered to represent different real-world objects.
Cooperating local schemata can be seen as a special case for contextual reasoning based on local model semantics, see [GG00]. From this point of view schema evolution (or schema integration) aims at finding a compatibility relation between schema revisions (or local schemata.) In the following, we will discuss aspects of evolution aware content management with the questionnaire application as an example.

5.1 Mappings Between Layers

The revision aware content warehouse is built on a layering model as introduced earlier. A schema is a tuple $\mathfrak{S} = (S, C, L, P)$ where S represents a meta-conceptual model, C a conceptual model, L a logical model, and P a physical model. On the meta-concept layer we use our notion of facets as defined in section 3.2 to express the meta-conceptual model: $S = \{\mathfrak{F}_1, ..., \mathfrak{F}_n\}$. For the conceptual model we use HERM syntax and semantics as defined in [Tha00].
The logical and physical layer may be defined by using a relational schema together with the computational support of a relational database management system.
We will assume that evolution will take place on the conceptual layer. Before we can talk about evolution, we have to connect the layers of the schema by appropriate mapping functions.

Meta-Concept Layer and Conceptual Layer are connected by associating a conceptual subschema to each facet on the meta-concept layer. We do not force the subschemata to be distinct or to cover the whole conceptual schema. Table 5.1 shows the intuition behind the facets of our questionnaire application.
Each concept is backed by specific characteristics, e.g. the name and the address for hunters, location information for districts, counting methods, details for species, etc.

The conceptual schema is formed by all concepts for all facets together with possible additional concepts that are not reflected in the meta-conceptual schema. These additional concepts are necessary to express integrity constraints between concepts of different facets, e.g. that rabbits will not live in foxes' dens. Comparing to the classical notions in relational databases, facets correspond to views over the conceptual schema. Therefore we define a total mapping function $\mathcal{T}^{S \to C} : \{\mathfrak{F}_1, ..., \mathfrak{F}_n\} \longrightarrow 2^{\{\mathfrak{C}_1, ..., \mathfrak{C}_m\}}$ between the meta-conceptual schema $\{\mathfrak{F}_1, ..., \mathfrak{F}_n\}$ and the conceptual schema $\{\mathfrak{C}_1, ..., \mathfrak{C}_m\}$. Each facet instance $\mathfrak{f}_i$ is mapped to a relationship that satisfies the subschema of the facet and works as an "entry point" on the conceptual layer.

Conceptual Layer. Logical Layer, and Physical Layer are connected by the usual mapping procedures known from textbooks, see e.g. [Tha00] and [HR01]. Analogously to the mapping between the meta-conceptual layer and the conceptual layer we will denote these mappings with $\mathcal{T}^{C \to L}$ and $\mathcal{T}^{L \to P}$. Transitive mappings will be denoted in the same manner. Because manual interaction will take place on the conceptual layer and the scope of the application will not call for extra manual tuning, we will consider the these mappings to be fixed and exclude the logical as well as the physical layer from the subsequent discussion.

5.2 Schema Revisions and Translations

Talking about schema evolution on the conceptual layer implies multiple conceptual schemata which are concurrently connected to the same meta-conceptual schema, figure 5.1 depicts the situation.
In this example we have two (simplified) conceptual schemata representing observations. For the subsequent discussion, we assume the following situation:

- Schema revision 1 was used for observations in the year 2002, so each object with revision 1 as home revision facilitates a $Time$ entity with a *year* property equal to 2002. In 2002 data about foxes and crows was raised. Schema revision 2 was used in the year 2003. In this year, only foxes were observed.

- Schema revision 2 was obtained from schema revision 1 by applying the following changes:

  - The property *name* of $Person$ was derived from the properties $Firstname$ and $Lastname$. Every person in revision 2 has a *name* property consisting of a firstname and a lastname.
  - The observation method in revision 1 was determined by a property $Kind$ with a domain of $\{hunting, counting\}$. In schema revision 2 there are relationships that express the observation method.

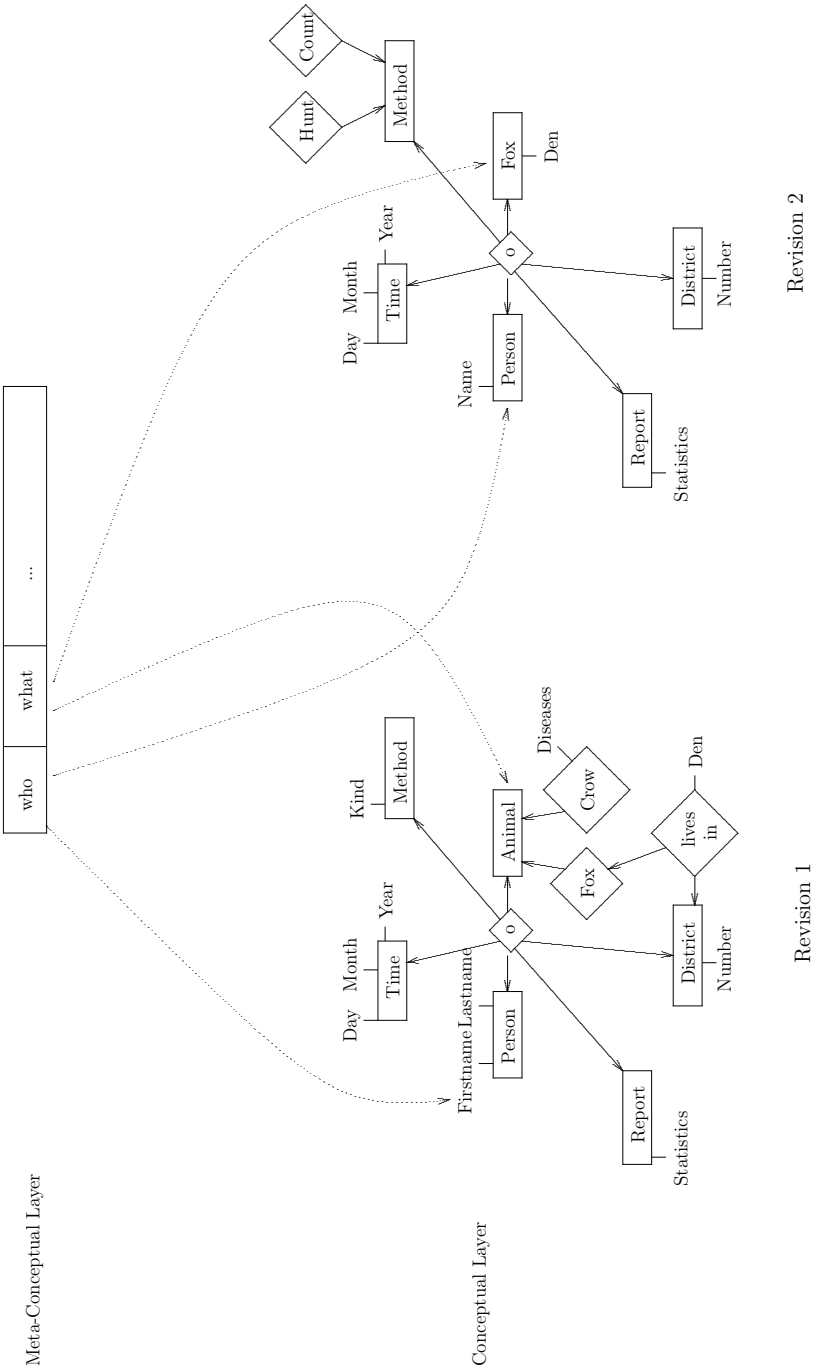

Figure 5.1: Schema Revisions

– The *Animal* section was redesigned, because we only consider foxes in schema revision 2. Objects that represent observations of crows are not relevant in schema revision 2.

Each conceptual schema C_i expresses the facets of the meta-conceptual layer in a specific way. In the following we will call a tuple $(C_i, \mathcal{T}^{S \to C})$ with a conceptual schema C_i and a facet concept mapping $\mathcal{T}^{S \to C}$ a *schema revision.*
A *revision aware content warehouse* is given by the tuple $(S, \mathcal{P}, \mathcal{V}, h, r, t)$ with

- a meta-conceptual schema $\mathcal{S}$
- a set of (global) property functions $\mathcal{P}$
- a set of schema revisions $\mathcal{V}$
- a head revision $h \in \mathbb{N}$
- a total, surjective, and injective function $r : \{1, ..., head\} \longrightarrow \mathcal{V}$ mapping revision numbers to schema revisions
- a partial translation function $t : \mathcal{V} \times \mathcal{V} \to Op$ which describes the differences between two revisions based on a set of (basic or complex) actions Op.

Please note, that properties are defined globally. Whenever a property p occurs in a schema, the same semantics is associated with this property.
The translation function between revisions is the base for automating mappings between different schemata within the content warehouse. New schema revisions are usually not created independently from each other. Often an existing schema revision is modified to reflect changes in perception of the represented world. These changes may be tracked by schema modification operations from the set Op. Operations are built from a set of base operations. Each base operation replaces a concept $\mathfrak{C}_1$ by a concept $\mathfrak{C}_2$ such that $\mathfrak{C}_1 \equiv \mathfrak{C}_2$ except the difference that is explicitly given:

- *adding named concepts*
- *removing named concepts*
- *adding a new property to a concept with/without default value*
- *extending the domain of a property*
- *restricting the domain of a property*
- *removing a property from a concept*
- *deriving a new property out of a set of old properties*
- *adding roles to a relationship concept*

- *removing roles from a relationship concept*
- *change relationships (e.g. relationships become entities)*
- *properties become relationships or vice versa*
- *change cardinality constraints for relationship concepts*
- *change local constraints like concept keys*
- *general cooperation functions mapping elements between concepts*
- Each *sequence* of operations is also considered to be an operation. These sequences are seen to be atomic from the point of view of the schema revision transition function.

The list of operations may be extended to suit current needs of an application. The translation function naturally includes merging of schema revisions S_1, S_2 to a new common schema revision $\hat{S}$, although schema merging needs manual conflict resolution.

5.3 Change Propagation

The state of a revision aware content warehouse is given by a set of objects $o = (id, \mathfrak{f}_1, ..., \mathfrak{f}_n)$. Each object o is created using its *home schema* revision (in the following denoted by $h(o)$), so the object's structuring on the conceptual layer is described by $T^{S \to C}(h(o))$.
Considering two schema revisions S_1, S_2 where S_1 is the home schema revision of object o we are interested in populating object o to revision S_2. We know that o is a model of S_1 and we also know that there are relationships between schema revisions S_1, S_2, so we can create a view o' on o that corresponds to the schema revision S_2 in the following cases:

1. The translation function t is defined for S_1, S_2: $t(S_1, S_2) = op$ with the operation op:
   a) Adding or removing concepts does actually not affect objects.
   b) If a new property was added with default value, we add this property to o' together with the given default value. If there is already such a property, we use the value that is present in o. If there is no default value, but the property is optional, o is left as it is. If there is no default value, but the property is mandatory, a propagation of o to o' is not possible.
   c) If a property is derived from other properties, the new property is calculated on o'. Propagation is always possible in this case.

d) If the domain of a property is extended, propagation is possible. If the domain of a property is restricted, a transition is only possible, if the value of the property in o is within the new domain.

e) If a property is removed, the property is removed from the view o'. Please note, that the property is only "hidden" (because it is removed from o'). A sequence that removes a property p and adds a new property p results in the original value of the property of o, not in the default value or a non-propagation result.

f) Adding a role to a relationship concept works like adding a property to a concept. If there is a default value or the role is already present, propagation is possible. Otherwise, propagation is not possible.

g) Removing roles from relationship concepts works like removing a property from a concept.

h) If the operation includes the change of local constraints, propagation is possible, if the new constraints are still fulfilled.

i) Propagation by general cooperation functions is possible, if the cooperation function is defined for object o.

j) The other cases are treated appropriately.

k) Sequences of operations are processed in the given order.

2. The translation function t is not defined for S_1, S_2, but for S_2, S_1: $t^{-1}(S_1, S_2) = t(S_2, S_1) = op$. We have to check whether the operation op is reversible:

   a) Adding a concept is undone by removing a concept. Removing a concept is undone by adding the concept (the concept's definition is present because all schema revisions are known.)

   b) Adding a property is undone by removing the property.

   c) A derived property is undone by reverting the calculation. If the inverse function is not available, a propagation is not possible.

   d) An extended domain is restricted, a restricted domain is extended.

   e) Removing a property is undone by adding the property.

   f) Adding / removing roles is undone by removing / adding roles.

   g) If a general purpose cooperation function was used, propagation is possible, if the inverse function is known and defined for o.

   h) Sequences of operations are processed in reverse order.

3. Neither the translation function t nor the inverse translation function are defined for S_1, S_2. But there is a sequence of schema revisions $\tilde{S}_i, ..., \tilde{S}_{i+k}$ such that $(t \cup t^{-1})(S_1, \tilde{S}_i) = op_1$, $(t \cup t^{-1})(\tilde{S}_i, \tilde{S}_{i+1}) = op_2$, ..., $(t \cup t^{-1})(\tilde{S}_{i+k}, S_2) = op_{k+1}$. The propagation from S_1 to S_2 is done along this path by applying the sequence of operations $op_1; ...; op_{k+1}$.

4. More generally, there is a number of paths connecting schema revisions S_1 and S_2 where propagation is possible. This situation occurs if merging of multiple schema revisions into one schema revision is allowed. Propagation possibly depends on the chosen revision path. For example, different revisions might add the same property, but with different default values or domains. This should not happen in a well-thought design, but it cannot be fully excluded from the discussion. Per definition, the shortest path between S_1 and S_2 is chosen. If there are multiple paths with the smallest length, the shortest resulting sequence of base operations may be chosen. If no deterministic choice can be made, propagation is not possible.

Each transition can be guarded by additional assumptions, e.g. an explicit "closed world" declaration: each object that cannot be propagated, does not contribute to the target schema revision. The next section will discuss this fact in more detail.
In our example there are objects created under revision 1: each object facilitates a $Person$ entity with a firstname and a lastname (e.g. an entity with $firstname =' John'$ and $lastname =' Smith'$), each object facilitates an $Animal$ which can be a fox or a crow and so on.
Additionally, there are objects created under revision 2 with the specified characteristics. Because we know that schema revision 2 was obtained by transforming schema revision 1, we also know that the object with the person entity has a representation in schema revision 2 with a name attribute for the personal part that equals to $(John, Smith)$. An object that facilitates an $Animal$ part for crows cannot be propagated, but we know from the translation that these objects will not contribute to queries in revision 2.

5.4 Querying Objects In A Revision-Aware Content Management System

Querying objects based on their home schema revision works in the classical way. But the user sees the content warehouse as a whole, every object should be accessible from every schema revision where it can be propagated to. A general query frame can be established:

1. The user chooses a "target" schema revision S. If no schema revision is explicitly chosen, the head revision becomes the default one.

2. The user formulates the query based on the concepts in the chosen schema revision S.

3. Based on the formulated query the system decides which schema revisions might be home revisions for relevant objects.

4. Based on the revision graph the system checks whether all possibly relevant objects can be propagated to the target schema revision. If this is the case, the query is evaluated on the objects of the target schema revision and all propagated objects from the relevant revisions. If it is not possible to propagate all possibly relevant objects, the query cannot be answered.

Queries in a content warehouse are formulated based on the structure defined on the meta-conceptual layer. A query $q = (q_1, ..., q_n)$ contains a subquery q_i for every facet $\mathfrak{F}_i$ in the meta-conceptual schema. The subqueries q_i can be formulated in an appropriate query language, e.g. SQL or Query By Example. The answer to query q is the set of all objects $o = (id, \mathfrak{f}_1, ..., \mathfrak{f}_n)$ whose facet instances satisfy the subqueries q_i.
Answers are based on the whole stock of objects available in the content warehouse, not only on the objects available in the target schema revision. To answer the query we need a representation of each object o in the target schema revision. We can distingiush the following cases:

1. The home schema revision of object o is connected to the target schema revision and o can be propagated to the target schema revision. In this case, there exists a representation of o in the target schema revision.

2. The home schema revision of object o is connected to the target schema revision, but o can not be propagated, because the propagated representation of o will violate integrity constraints during propagation.

3. The home schema revision of object o is not connected to the target schema revision. Propagation of object o to the target schema revision is not possible at all.

The notion available in the target schema revision can be refined by the following: the answer to query q relies on all objects, that

1. are located in the target schema revision (the target schema revision is the home schema revision) and therefore satisfy the target schema revision,

2. are not located in the target schema revision but satisfy it, or

3. can be propagated to the target schema revision.

Queries to the content warehouse can always be answered. The interesting point is the validity of the answer to a query in relation to the objects in the content warehouse. Objects that cannot be propagated to the target schema revision are excluded from query evaluation. If an object o was excluded from evaluation, we can generally distinguish the following two cases:

1. o carries parts of the information that was requested by the query. The query's answer delivered by the system is incomplete.

2. o does not contribute to the query's answer. The query's answer is complete.

Unfortunately, it is not possible to distinguish between these two cases in general. For practical applications it is usually sufficient to estimate completeness of the query's answer:

- The home schema revision of o is connected to the target schema revision, but o was not propagated because it violates integrity constraints during propagation. The integrity constraints can be relaxed during propagation. The result of this "relaxed propagation" is a representation of o that partially satisfies the target schema revision. This partial information can be seen under a possible world semantics: if it is possible to complete this partial representation of o such that

  1. o satisfies the target schema revision and
  2. o contributes to the query's answer

  the query's answer is considered to be incomplete. A full discussion of partial information (in relational databases) can be found in [Kle97].

- A similar (or more general) discussion can be found for objects o in schema revisions that are not connected to the target schema revision. If it is possible to complete o such that o satisfies the target schema revision and o contributes to the query's answer, the answer is considered to be incomplete.

Estimation of completeness is based on the assumption, that an object is relevant for the query's answer until it is obvious that it does not contribute to the result. This decision will be made based on the concept's characteristics of the query and the schema revisions that contain the objects under investigation.
Each subquery q_i will naturally refer to concepts $\mathfrak{Q}_{i,1}, ..., \mathfrak{Q}_{i,m}$ to express the subquery's answer. For each concept $\mathfrak{Q}_{i,j}$ a weaker concept $\tilde{\mathfrak{Q}}_{i,j}$ can be found, that expresses only mandatory or optional properties of elements of concept $\mathfrak{Q}_{i,j}$. $\tilde{\mathfrak{Q}}_{i,j}$ has to consider derived properties and properties that were used to derive new properties, too: if a derived property p is present in $\tilde{\mathfrak{Q}}_{i,j}$ and p was (potentially transitive) derived from $p_1, ..., p_k$, then $p_1, ..., p_k$ are relevant to $\tilde{\mathfrak{Q}}_{i,j}$. On the other hand, if p is part of the characteristics of $\tilde{\mathfrak{Q}}_{i,j}$ and $\hat{p}$ is derived from p, then $\hat{p}$ is also relevant to $\tilde{\mathfrak{Q}}_{i,j}$. For each subquery a set $\{p_{q_i,1}, ..., p_{q_i,l}\}$ of relevant properties can be found. These relevant properties form a query property space $\mathbb{Q}_i$. In this space we identify bounding boxes $[(p_1^{min_1}, p_1^{max_1}, ..., p_l^{min_1}, p_l^{max_1}]$, ..., $[(p_1^{min_m}, p_1^{max_m}, ..., p_l^{min_m}, p_l^{max_m}]$ that characterize the restriction of used values from the domains of the relevant properties. For example, if the query references objects with $2002 \leq year \leq 2004$ for a property $year$, we know that objects with property $year = 2005$ will not contribute to the subquery's answer. If no restriction can be made for a relevant property, the whole domain is considered to be relevant.

Analogously to the query property spaces we can derive schema revision property spaces $\mathbb{S}_i$: the set of relevant properties is given by all properties present in any object under investigation. Objects under investigation are those objects that cannot be propagated and do not satisfy the target schema revision. The values of these properties for the objects in the schema revision define the bounding boxes within $\mathbb{S}_i$.
Based on query property spaces and revision property spaces we can decide which of the "problematic" revisions might contribute objects to the query's answer: a schema revision might contribute, if there is no subquery q_i such that the bounding boxes of q_i do not overlap with the bounding boxes of schema revision i.

6 Architecture

In this chapter we will discuss aspects of implementing the conceptual framework that was introduced in the last chapter. We will analyse how a revision-aware content warehouse can be mapped to storage structures on the physical layer. But talking about content also means talking about delivery of content to the user. We will discuss how user interaction can be modeled to adapt the user interface to conceptual content changes.

6.1 Deriving Storage Structures

The classical approach of defining an all-embracing logical schema which determines all facets of the application directly supports deriving of storage structures. Because tuples of relations in the relational model (and similarly entities and relationships in the entity relationship model) share a common and simple structure efficient access can be achieved by access paths. This approach cannot be directly applied in the case of a revision-aware content warehouse. There is no rigid structural description for objects, only a set of schema revisions that provide basic, only partial, and maybe contradictory information of object structure.
Because user interaction should not take place below the conceptual layer, the system has to derive appropriate structures. In modern content management systems time is usually more expensive than space and read access usually occurs much more often than updates. So the algorithm is driven for optimizing read access time. Content management systems are usually closed: access is only granted if user interfaces under system control are used. So we have full freedom of dynamically adapting the storage structures, we are free to facilitate controlled redundancy or controlled relaxed integrity constraints.
Like in any other system for storing persistent data we have to consider different cases of accessing data:

1. *Access by reference or primary key* facilitates the identification mechanism used in the content warehouse. Identification can be system controlled (system identificator, reference) or based on the object's values (key based). This (primary) access path is the backbone of every data storing system. Modern (embedded) storage engines offer APIs that are capable of storing objects of arbitrary structure in form of $(key, value)$ pairs where key represents a globally unique identifier and $value$ contains arbitrary, system uninterpreted data. The $(key, value)$ dictionaries are implemented using efficient and highly scaleable B*-trees or hash tables.

2. *Access by values* supports access to objects based on the values of arbitrary properties. Access by values does not rely on uniqueness of the result. To access property values the system has to interpret the *value* part of the dictionary which makes use of the primary access path inefficient. The whole stock of data has to be "scanned" for qualifying objects. This applies to every situation independently from the underlying data model. In the case of arbitrary object structures this is especially expensive because each object has to be touched and unmarshalled for its own.

Storing and accessing of objects with arbitrary structure was intensivly studied in the area of semistructured database management systems, see e.g. [MAG+97]. These approaches use general purpose graph models to express the object's structure together with its properties and values. Accessing an object means traversing the object graph. This approach facilities generality at the expense of efficiency. That's why indexing structures were developed ([MWA+98]) which perform well under pragmatical assumptions.
In our situation we can restrict the general case because schema information is partially available. We have a set of schema revisions which define a conception of the objects within this revision. In difference we cannot use a one-to-one mapping between concepts and storage units because objects may be represented by multiple concepts. Additionally, objects may be represented differently in different schema revisions.
The question of schema revisions can be solved in two ways:

1. Each schema revision is seen as an independent storage schema within the content warehouse. Query processing in this configuration can be adapted from query processing in distributed database systems.

2. All schema revisions share a common storage schema. This method is prefered if schema revisions share parts of their structures. This assumption can be made because definition of schema revisions relies on former revisions. So a partial overlapping between revisions will naturally occur.

Handling different representations of objects in different revisions results in the classical decision whether controlled redundancy should be incorporated or not:

1. Each object is stored in its home schema revision. Propagation is done at query evaluation time. Each schema revision has to be checked whether there are objects that satisfy the target schema revision. This is the space optimized version.

2. Each object is stored in its home schema revision. In each schema revision that is satisfied by the object, a "proxy object" is installed which links to the original object. During query evaluation time only the target schema revision has to be checked. If a proxy object is found the home schema revision is

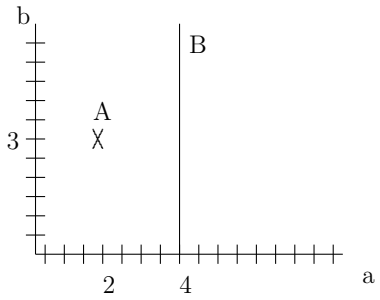

Figure 6.1: Representation of Objects in the Property Space

consulted and the object is propagated. This is a trade-off between time and space. Updates need a recalculation whether the object is still a model of the schema revision or not.

3. Each object is stored in its home revision. In each schema revision that is satisfied by the object a "proxy object" is stored together with the object's representation in this schema revision[1]. Only the target schema revision has to be consulted at query evaluation time. When updates occur, the update is made in the home revision together with all revisions where a representation exists. This is the time optimized version. Because time is more expensive than space and read access occurs much more often than updates[2] this is the preferred configuration.

As a third problem the representation of objects within a schema revision (or a set of schema revisions if we incorporate multiple schema revisions in one storage structure) remains. Concepts explicitly describe the object structure only partially and may overlap. For a value oriented access we need explicitly accessible object properties although it is not possible to maintain an access path for every combination of properties because the number of possible combinations grows exponentially with the number of properties. Multidimensional access structures like Gridfiles or R-trees address this problem ([HR01]).
But nevertheless, also these multidimensional access paths rely on fixed structures. First we have to find such a structure that approximates our irregular objects. A first naive approach may be the revision property space introduced in section 5.4. If we consider every property that is used within any object of the schema revision, this spans a high-dimensional space. Because objects will not be determined in a possible world sense in every property dimension of this space each object is represented by a subspace, not only by a single point. Figure 6.1 depicts the situation in the two-dimensional case for two objects $A = (a : 2, b : 3)$ and $B = (a : 4)$.

[1] If multiple revisions share the same representation, this could be compressed.
[2] In the questionnaire example application — like in any other data warehouse application — updates only occur when new data is loaded and in the case of error correction.

Querys are mapped to this space by performing intersection operations between hypercubes defined by the query and the objects in the space. But unfortunately, this naive approach lacks of one problem: multidimensional access structures perform badly on higher dimensions because the approximation by intersection operations of bounding boxes incorporates areas that are not relevant. In higher dimensions the volume of these areas increases significantly which adds additional overhead. To keep the number of dimensions and therefore the number of properties as small as possible is necessary for an efficient access.
As the general idea we define three types of storage structures for value based access:

1. The first kind of structure is a multidimensional property space where each object is represented as a point. This means that the values of all properties in this space are determined for every object. This allows simplified query operations because checking for intersection is reduced to checking for containment. We will call the corresponding concept the *kernel concept*.

2. The second kind of structure is a multidimensional property space where each object is represented within a subspace. These property spaces should be chosen such that the dimension of the subspaces is as low as possible. The corresponding concept will be called *enhancement concept*.

3. The third kind of structure is a general purpose storage which supports maximum flexibility in structure but only scanning the object as access to the object's properties. The corresponding concept will be called *cargo concept*.

To identify the property spaces for storage types 1 and 2 we have to find appropriate sets of concepts $\mathcal{C}^1$, $\mathcal{C}^2$, $\mathcal{C}^3$ such that each object can be losslessly decomposed in objects o_1, o_2, o_3 such that o_i satisfies at least[3] one concept in $\mathcal{C}_i$. To find proper concepts for this purpose formal concept analysis based on concept lattices ([GW98]) can be used to structure objects and properties in the schema revision.
The fourth problem is the discussion of materializing concepts in a schema revision. A materialized concept works like a view in traditional database systems: accessing the concept's extension is not performed by evaluating the concept's characteristics on the stock of objects but by accessing the set of objects directly. Two cases have to be distinguished, they determine on which level materialized concepts are supported:

1. *Extension scan:* the concept's extension should be simply scanned. This is mainly a separation on the level of the primary access path. Either there exist multiple access paths or concept containment is incorporated by stub records, e.g. following the approach that realizes the PostgreSQL rule system ([SHH86]).

[3] At best, each object o_i satisfies exactly one concept from $\mathcal{C}_i$, otherwise additional redundancy is necessary

2. *Access by concept and value:* Additional to the value based access as described above defined (named) concepts are incorporated in queries. This case can be treated in the following ways:

   a) For each concept that should be materialized an access by value path as described above is installed.

   b) Concept containment is seen as a property of an object. For each concept to be materialized, a new boolean property is introduced indicating whether the object satisfies the concept or not. This property space could be separated from the ordinary properties or incorporated into the three storage types.

The decision which concepts are materialized depends on the usage profile of the content. Concepts defined over an explicit extension are naturally materialized. Additionally, the system has to keep statistics about importance of certain concepts and properties in relation to the actual queries. The base for the decision is the conceptual modeling of the user interface of the content management system as described in section 6.2 which identifies relevance of queries at least partially. At certain points of time statistics can be evaluated to identify changed preferences in concept or property access and to adapt the storage structure. Materialized concepts that are accessed less frequently can be virtualized, concepts that are accessed more frequently are materialized. Properties that are used more often moved in the property spaces towards the kernel concept, less frequently used properties are moved towards the cargo concept.

6.2 User Interface Maintenance

Content management system do not only consider persistent storage of data but also the delivery of content to the user. Content systems are usually closed systems: content is only delivered via interfaces that are known to the system.
Information about the flow of content between the system and the user can be incorporated in adaptation decisions, e.g. as explained in section 6.1. In this section we will discuss how we can conceptually represent user interaction in our content system.

6.2.1 Modeling Interaction

Interaction between a user and a system can be modeled as a dialog where each participant (system and user) changes between the roles “speaker” and “listener”. This dialog can be expressed from the point of view from one of the participants. In this case we have to model a bidirectional reactive system. Usually it is sufficient to assume that the system as the “non creative” part will react on the users actions. We will use the specification language *Sitelang* ([TD01]) to express the dialog between

user and system. Sitelang is based on the concepts of storyboarding and offers constructs to express scenes and scenarios. The basic interaction units are dialog steps: the user triggeres actions based on content that was offered by the system. For incorporating Sitelang in Web applications based on the Model View Controller paradigm see e.g. [FCF+06].

6.2.2 Structuring Dialogs

The Sitelang framework expresses dependencies based on the dialog flow. At this point we will concentrate on the ways to express the content that is delivered to the user as a decision base for triggering dialog steps.
If we analyse a dialog that is e.g. displayed in a Web browser, on a rich client, or is delivered via email or PDF printout, we can identify the following components:

- *Static content* like the company's or institution's logo or the application name. We also consider statically linked dynamic elements like advertisement to be static content from the point of view of the system. Static content is present on each dialog and works as pure decoration. From the application's point of view there is no semantics associated with static content.

- *Decorative content* depends on parameters of the dialog but it has no meaning to the application like the dialog title, contextual help, or integrated services like contextual weather forecast or contextual advertisement. Decorative content can be statically on different levels: equal for every dialog of a certain user, equal for a certain part of the story, equal during a user session, or different on each dialog.

- *Delivered content* depends on the content objects that were accessed during the scenario. Typical examples are query or browsing results.

- *Events with/without escort data* allow the user to trigger new dialog steps. These components are usually annotated, e.g. labels for input controls or hints.

Dialog structures can be delivered using different interaction channels. For example, acquisition of observation results in our example application may be done by PDF printouts, online using a conventional browser, or a mobile device. That's why the creation process of dialog documents is divided into several phases, also called the "onion approach" ([Tha03]):

1. Delivered content and events are identified from the Sitelang specification. We are using Topic Maps ([Top06]) as a general purpose facility to express these structures. Occurences of subjects within this topic map can be formulated as parameterized queries where the values for the query's parameters are obtained from the content which is associated to this topic map during delivery.

2. The topic map obtained in step 1 is contextualized: information based on the user's profile and portfolio is added, e.g. localized labels, hints or explanations of different granularity. Contextualizing can be made based on different contexts, e.g. user, time, place, or delivery channel ([KSTF04]).

3. Based on the delivery channel the general layout is defined: the annotated topic map from step 2 is combined with static content. Multiple topic maps may be composed for multi-modal channels, e.g. frame-like websites.

4. Based on the layouted topic map a concrete description of the optical properties are determined. This includes definition of visual styles. The result of this step is a description a renderer can interpret, e.g. a HTML document, a PDF description based on formatting objects, or a SVG document.

5. The renderer creates the physical representation which is visualized and presented to the user. HTML documents may be shown in a Web browser, formatting objects documents may be rendered as PDF and printed out, SVG documents may be rendered as a RGB image displayed in an image viewer.

This rendering pipeline is usually implemented using XML Stylesheet Transformations (XSLT) with a device specific backend for the last step. XSL as a declarative language requires great amounts of system resources to perform transformations. Adding XSLT support easily multiplies response time.
Fortunately the number of parameters that cause different rendering results is usually limited within an application: the user's profile and portfolio can be approximated by actor definitions or the number of different output formats is limited to a small amount. Together with the size of today's disk space it is easily possible to precalculate a representative set of dialog implementations in the target language like JSP, PHP, or other scripting languages. For example, if you have to support two different user profiles *expert* and *normal* which differ in the amount of presented contextual help, four different formats $HTML$, $PDF\ online$, $PDF\ printout$, and two different cooperate designs, you have to maintain $4 \cdot 2 \cdot 2 = 16$ versions for each dialog obtained from the Sitelang specification. Although the number of dialogs grows exponentially, disc space is a smaller problem than time. Together with a multi level cache strategy for static content a significant performance gain can be achieved (for a deeper discussion see e.g. [WV].)

6.3 Generating Applications: The Two-Stage Approach

A content management system incorporating conceptual facilities enables system support for various user tasks: the administrator is supported by an automatic storage management, the domain expert can directly design on the conceptual level, and the normal user benefits from a user-centric interaction and query modeling. But

many features also impact non functional requirements of the content management system: additional management of conceptual structures influences query processing time. Security is also affected: semantical information may be used in a way that was not in mind of the system's designer.
For that reason, content management systems use a two-stage approach: each system is divided in a full-featured *master system* which is used by the system's administrators or priviledged users and a *dependent live system* which serves the mass of all requests and is optimized for performance.
Applying this approach to our scenario we can identify the following tasks for the two stages:

- The master system provides the repository for managing the semantic information on the conceptual layer: revisions, properties, concepts, the revision graph. Additionally the master supplies all functionality that is necessary to process this data, e.g. query validity check, concept lattice analysis, statistical data, and ad hoc query facilities. The conceptual interaction model is also represented in the master's repository: topic maps, context defintions, conext annotation specifications, layouts, or delivery channel specification.

- Based on the information stored in the repository the master system derives storage structures, interactions elements (scripts) and query execution plans for all predefined queries used in the dialog topic maps. This step can be compared to the compilation of a programm out a set of source files. The result lacks of all information that is not necessary for serving user requests, but available optimization parameters are incorporated for the different cases. The live system reports back usage statistics that can be used by the master to dynamically adapt the generation process. If updates are allowed over the live system, this has also be reported to the master system. For security reasons, the live system can be used in flooding mode where the (access protected) master overrides any information on the live system to prevent hacks. Alternatively, deploying the live system can be made dependant on certain events on the master repository like creating a new revision or automatic adaptation.

Conclusion

This work presented an approach to conceptually handle data in content management systems. Based on a conceptual model that allows to combine structural and intensional aspects of information and that combines advantages of schema oriented and element oriented paradigms a representation of content at different layers of abstraction was presented. To handle content from different points of view cooperating schemas within a content management system were discussed. The schema cooperation was based on the notion of schema revisions: different representations of the same fact in different manners. Additional to these conceptual foundations, aspects of implementation were discussed such as an automatic generation of storage structures of flexible structured content and conceptual modeling of user interaction based on topic map representations of dialogs.

Bibliography

[Bea98] George Bealer. Intensional Entities. In E. Craig, editor, *Routledge Encyclopedia of Philosophy*, volume 4. Routledge, 1998.

[BLN86] Carlo Batini, Maurizio Lenzerini, and Sham B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[Con86] B. Convent. Unsolvable problems related to the view integration approach. In *ICDT'86*, volume 243 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 1986.

[DM00] Marie Duží and Pavel Materna. Constructions. `http://til.phil.muni.cz/text/constructions_duzi_materna.pdf`, 2000.

[EKMR99] H. Ehrig, H.J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, chapter Applications, Languages and Tools. World Scientific, Singapore, 1999.

[FCF+06] Gunar Fiedler, Andreas Czerniak, Dirk Fleischer, Heye Rumohr, Michael Spindler, and Bernhard Thalheim. Content Warehouses. Preprint 0605, Department of Computer Science, Kiel University, March 2006.

[Fit00] Melvin Fitting. Modality and Databases. In Roy Dyckhoff, editor, *TABLEAUX*, volume 1847 of *Lecture Notes in Computer Science*, pages 19–39. Springer, 2000.

[Fit04] Melvin Fitting. First-order intensional logic. *Ann. Pure Appl. Logic*, 127(1-3):171–193, 2004.

[Fit06] Melvin Fitting. Intensional Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Stanford University, Fall 2006.

[FRT05] Gunar Fiedler, Thomas Raak, and Bernhard Thalheim. Database Collaboration Instead of Integration. In Sven Hartmann and Markus Stumptner, editors, *APCCM*, volume 43 of *Conferences in Research and Practice in Information Technology*. Australian Computer Society Inc., 2005.

[FT04] Gunar Fiedler and Bernhard Thalheim. Towards Linguistic Foundations of Content Management. In Farid Meziane and Elisabeth Métais, editors, *Natural Language Processing and Information Systems*, volume 3136 of *Lecture Notes in Computer Science*, pages 348–353. Springer, 2004.

[GG00] C. Ghidini and F. Giunchiglia. Local Models Semantics, or Contextual Reasoning = Locality + Compatibility. http://citeseer.ist.psu.edu/481285.html, April 2000.

[GW97] Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 436–445. Morgan Kaufmann, 1997.

[GW98] B. Ganter and R. Wille. *Formal concept analysis - Mathematical foundations*. Springer, Berlin, 1998.

[HR01] Theo Härder and Erhard Rahm. *Datenbanksysteme: Konzepte und Techniken der Implementierung*. Springer Verlag, second edition, 2001.

[Kan96] Hannu Kangassalo. Conceptual Description for Information Modelling Based on Intensional Containment Relation. In Franz Baader, Martin Buchheit, Manfred A. Jeusfeld, and Werner Nutt, editors, *KRDB*, volume 4 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1996.

[Kau67] Raili Kauppi. Einführung in die Theorie der Begriffssysteme. Technical Report Ser. A, Vol. 416, University of Tampere, 1967.

[Kle97] Hans-Joachim Klein. Gesicherte und mögliche Antworten auf Anfragen an relationale Datenbanken mit partiellen Relationen. Habilitation Thesis 9802, Christian-Albrechts-Universität Kiel, 1997.

[KSTF04] Roland Kaschek, Klaus-Dieter Schewe, Bernhard Thalheim, and Gunar Fiedler. Contextualizing Electronic Learning Systems. In Kinshuk, Chee-Kit Looi, Erkki Sutinen, Demetrios G. Sampson, Ignacio Aedo, Lorna Uden, and Esko Kähkönen, editors, *ICALT*. IEEE Computer Society, 2004.

[MAG+97] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, and Jennifer Widom. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):54–66, 1997.

[MWA+98] J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajamaran. Indexing semistructured data, 1998.

[Nii04] Markopekka Niinimäki. *Conceptual Modelling Languages*. PhD thesis, University of Tampere, 2004.

[Pal94] Jari Palomäki. *From Concepts to Concept Theory: Discoveries, Connections, and Results*. PhD thesis, University of Tampere, 1994.

[Pei58] Charles Sanders Peirce. *Collected Writings*, volume 1 - 8. Harvard University Press, Cambridge, MA., 1931-1958.

[Rod95] John F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393, 1995.

[SHH86] Michael Stonebraker, E. Hanson, and C-H. Hong. The Design of the POSTGRES Rules System. Technical Report UCB/ERL M86/80, EECS Department, University of California, Berkeley, 1986.

[TD01] B. Thalheim and A. Düsterhöft. SiteLang: Conceptual modeling of internet sites. In *Proc. ER'01, LNCS 2224, Springer, Berlin*, pages 179–192, 2001.

[Tha00] B. Thalheim. *Entity-relationship modeling – Foundations of database technology*. Springer, Berlin, 2000. See also http://www.informatik.tu-cottbus.de/∼thalheim/HERM.htm.

[Tha01] B. Thalheim. Abstraction layers in database structuring: The star, snowflake and hierarchical structuring. Technical Report Preprint I-13-2001, Brandenburg University of Technology at Cottbus, Institute of Computer Science, 2001. See also: http://www.informatik.tu-cottbus.de/∼thalheim/slides.htm.

[Tha03] B. Thalheim. Co-Design of Structuring, Funtionality, Distribution, and Interactivity of Large Information Systems. Coumputer Science Reports 15/03, Cottbus University of Technology, Computer Science Institute, 2003.

[Tha05] Bernhard Thalheim. Component Development and Construction for Database Design. *Data & Knowledge Engineering*, 54(1):77–95, July 2005.

[Top06] TopicMaps.org. XML Topic Maps. http://www.topicmaps.org/xtm/, Sep 2006.

[W3C04a] W3C. Document Object Model (DOM) Recommendation. http://www.w3.org/DOM/, Jan 2004.

[W3C04b] W3C. XML Schema Part 0: Primer Second Edition. http://www.w3.org/TR/xmlschema-0, Oct 2004.

[WV] Johannes Willkomm and Markus Voss. Delivery Architectures von Informationsportalen, High Performance unter Hochlast. `http://www.sdm.de/web4archiv/objects/download/fachartikel/1/cm_willkomm_voss.pdf`.

Acknowledgements

We like to thank our students Anatol Frick, Max Fritzsche, Ariane Nouidui-Tchagou, Tsvetelin Polomski, and Ove Sörensen for investigating implementational aspects of some ideas in this work. We also thank Heiko Schmüser from the Ecology Center at Kiel University and the Hunting Association of Schleswig-Holstein (Landesjagdverband Schleswig-Holstein e.V.) for providing the case study used in this work.