Application Experience with a Repository System for Information Systems Development*

Manfred A. Jeusfeld[1], Matthias Jarke[2], Martin Staudt[3]
Christoph Quix[2], and Thomas List[2]

[1] INFOLAB, Tilburg University, 5000 LE Tilburg, The Netherlands
[2] RWTH Aachen, Informatik V, 52056 Aachen, Germany
[3] Swiss Life, CH/IFuE, Postfach, CH-8022 Zürich, Switzerland

Abstract. The purpose of a computerized information system is to improve data-intensive business processes in an organization. The development of such systems itself is data-intensive. Tools have been developed to support the development process, among others repository systems which serve as the common database used by the development team. This paper reports on application experience with the ConceptBase system which we developed over the last 12 years. Eight applications are presented in more detail to show which properties the users demanded and how different applications used the base functionalities of the system in a different way. In summary, extensibility, performance, and advanced query facilities turned out to be most important.

1 Introduction

In the late seventies, database researchers felt the need for a design language which not only covers the static aspects of a database, i.e. its schema, but also dynamic aspects, e.g. transactions. A well-known example is the Taxis design language [19]. Following the Taxis idea, corresponding efforts were made for the requirements specification level resulting in the requirements modeling language RML [7, 6] for information systems. The knowledge representation language Telos [18] was later introduced as a generalization of RML putting emphasis on extensibility.

We started the development of ConceptBase in 1987 as part of the Esprit project DAIDA [9]. The system was originally intended as a repository for design objects created through the development of information systems (IS) [14]. To do so, the system had to be able to represent objects expressed in rather heterogeneous formalisms. We decided to join the development of the knowledge representation language Telos because of its flexible meta class mechanism and its advanced deductive rule language. As the use of the system in DAIDA was successful, the system was made available in 1990 to research institutes and some companies for their information systems projects. Each 18 months, a new release of the system was published based on the evolving requirements gathered from users of the previous release. Hence, ConceptBase has been extensivley used over the last ten years for different aspects of information systems development. This paper has the goal to present this experience to the IS developer community. It shall give some insight which mix of methods is successful and where methods have their limits.

The paper is organized as follows. First, the technologies implemented in ConceptBase are briefly introduced together with the motivation why they were included into the system. Then, we present case by case successful application projects and discuss how they combined the technologies to realize their goals. The application projects are classified into three categories:

* Correspondence should be directed to the first author at jeusfeld@kub.nl

– model integration: Such application projects require a customized collection of inter-related modeling languages. ConceptBase supports such projects by its *meta modeling* features.
– model analysis: Such applications have to handle large models whose content has to be analyzed to find hidden properties or faults. This task is mainly supported by the advanced *query language* of ConceptBase.
– distributed co-operation: Here, a group of human experts has the task to process highly inter-related information. ConceptBase provides facilities for automatic *view management & trading* based on Internet standards for such application domains.

Typically, an application project falls into more than one category. Therefore, an application is presented from the viewpoint of these three categories. This paper does not intend to compare the ConceptBase system with competing approaches (see [16] for more information) or to present technical details of the system (see [10]) or of the application projects (see respective references). Instead, we show how rather heterogeneous requirements from application projects were fulfilled by a rather small set of base technologies.

2 The Technology used in ConceptBase

ConceptBase is a client-server database system for conceptual information. The representation language for the conceptual information is Telos [18], originally a knowledge representation language for encoding requirements models for information systems. The foundation of Telos is a simple data structure which uniformly represents objects, classes, meta classes, attributes, and class-instance as well as subclass-superclass relationships. ConceptBase features a persistent main memory database for efficient storage and access of Telos objects. Telos also provides for a logical sublanguage to express integrity constraints, deductive rules, and queries. The latter were specifically designed for ConceptBase: queries are classes with a (logical) membership constraint. The logical sub-language provides predicates to access objects and their attributes plus some aggregation predicates like SUM and AVG. Deductive rules are used to define new predicates from existing ones. In ConceptBase, the logical sub-language is a variant of DATALOG with negation. The uniform representation of objects plus well-defined semantics of the logical sub-language [15] are the preconditions for the extensibility of the system for various application types.

Since the class-instance relationship is stored explicitly in ConceptBase, an object is allowed to have multiple classes. Moreover, a class may be instance of other classes, called meta classes. Meta classes may have classes called meta meta classes, and so on. There is virtually no limit in this hierarchy though most applications do not go beyond 4 levels (instances, classes, meta classes, meta meta classes). The lowest level can be associated with data, the class level corresponds to the schema level (of databases), the meta class level encodes modeling languages (e.g. the entity-relationship language), and the meta meta class level can be used to link multiple modeling languages. Integrity constraints, deductive rules and queries can be formulated at any of these levels. For example, integrity constraints at the meta class level can be used to define the semantics of cardinality tags. The logical sublanguage was designed to allow advanced analysis of stored models.

Figure 1 shows how objects of different abstraction levels can be represented in ConceptBase. The lowest level (instances) is used to store factual data, the next level contains classes (grouped into models). The classes of these classes (meta classes) constitute a modeling language. Finally, meta meta classes are used to define how modeling languages can be defined and linked with ConceptBase. In the example of the figure, a very simple meta meta model is used which just provides the facility to talk about nodes and their connections. More complex meta meta models are possible when more application knowledge has to be represented (see application examples below). Besides

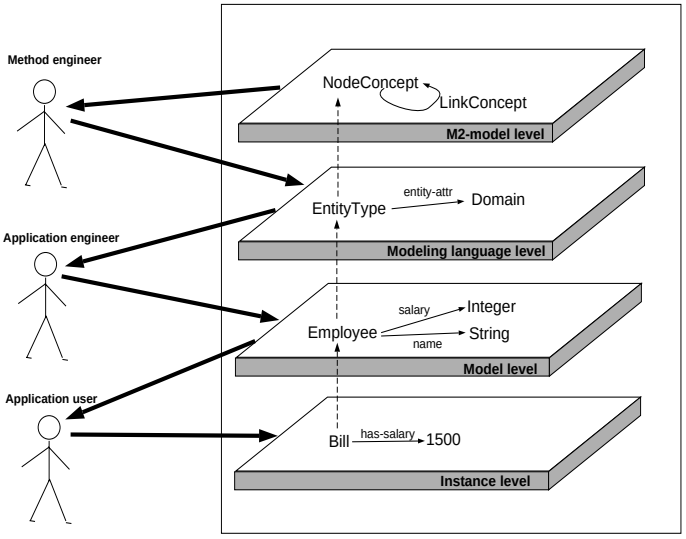

Fig. 1. Instances, models, modeling languages and meta meta models

the level are typical user groups of the system. A *method engineer* learns the meta meta classes and uses them to define a method[1]. An *application engineer* learns a method and creates models, for examples some ER schema. Finally, an application user is trained about application details and handles factual concepts (like data).

Besides the logical sublanguage, ConceptBase allows to define event-condition-action (ECA) rules where events are updates to the database or calls for query evaluations. The condition part is a logical expression (a query on the database). Actions may be updates to the database or calls of external routines. Active rules are more expressive than deductive rules at the cost of potential infinite loops. They were introduced to allow more flexible reactions to database updates. Before that, a reaction to an update was either to accept or reject it[2]. With user-defined active rules, one can for example specify a repair action when incomplete information is entered into the database.

The client-server architecture of ConceptBase allows clients to connect via the Internet to a ConceptBase server. The communication between client and server is established through sockets using formatted ASCII messages. This mechanism allows not only the usual way of client-server interaction (client request is followed by server response). The server can also send notification messages to client programs (e.g., user interfaces to ConceptBase) without a request of a client before. One application of notification messages is maintenance of externally materialized views. Another application is to inform the client programs of specific events in the database.

3 Experiences from Applications

Since 1989, the ConceptBase system has been freely available to research groups all over the world. About 250 installations have been reported to the ConceptBase development team so far. Thus,

[1] We use the term method here to refer to a collection of interrelated modeling and design languages. Structured analysis (Yourdan) is an example of a method.

[2] ConceptBase compiles integrity constraints into a set of active rules which specify for each update which condition to check. If the condition is true, then the action ‘accept’ is performed making the update persistent. If it’s false, then the database is set back to previous state.

the subsequent list is definitely not complete. It shall instead give an overview of characteristic application projects. All usages of ConceptBase reported below were undertaken by experts of their field with considerable background in modeling and logic. Both is required since ConceptBase is based on advanced modeling principles and on first-order logic for expressing rules, constraints, and queries.

3.1 Database application development

The pioneering work of the DAIDA project [9] influenced greatly the functionality of the ConceptBase system. DAIDA created methods and tools for developing data-intensive applications. The methods covered systems modeling (comparable to requirements engineering), application design, and application implementation. For all three levels, specialized notations (or languages) were developed. ConceptBase had the task to serve as a global repository which

- maintains the design objects, i.e. all artefacts created during application development regardless of their notation,
- interrelates design objects by design decisions, i.e. the dependencies between artefacts,
- communicates with the design tools to capture changes to design objects according to a software process model.

ConceptBase was used in two basic modes. In the *modeling mode*, a software process model [12] about design objects, design decisions, and design tools was formulated by appropriate meta classes. This includes the representation of the notations used within DAIDA. In the *operational mode*, ConceptBase was part of the development environment where the external tools ask queries (e.g. to fetch the current design objects) and report their results (e.g. the new design objects together with dependency links to existing design objects). Graphical tools for navigating the resulting dependency network were provided as part of the ConceptBase user interface. Constraints were used to prohibit tools from storing incomplete or inconsistent design objects in the repository. Queries were mainly used to determine applicable design decisions for a given design object (referring to the software process model which was part of the design database).

Discussion. DAIDA was one of the first projects that made a formal software process model the basis for its development methodology and the basis for the global design repository. The idea to establish traceability among artifacts through such a software process model was refined in the REMAP project between New York University and the Naval Postgraduate School, Monterey [24]. The ConceptBase metamodels developed in REMAP have meanwhile served as a blue print for a number of commercial traceability tools, e.g. by Andersen Consulting and by Texas Instruments and formed one starting point for our research on process integrated tools [23, 3]. ConceptBase lacked at that time a flexible query language. Moreover, performance of the integrity checker was so low that an update to the repository took several minutes. Since all notations used in DAIDA were at least partly mirrored as ConceptBase meta classes, a change to a notation caused a major revision of the meta classes. The communication to external tools was limited to accessing and storing design objects as frames. Consistency checks did cross the notations and were fairly sophisticated. However, the tools who triggered the checks were not programmed to react intelligently to error messages from the ConceptBase repository. Nevertheless, DAIDA was an excellent testbed to learn the requirements for a repository system. Subsequently, applications of ConceptBase were more focused on fewer notations and fewer external tools. In the subsequent WibQuS [13], a repository was built that served as a semantic trader between tools for industrial quality management. Here, the tools were coupled to ConceptBase via commercial databases. ConceptBase was no longer used for tool communication. Instead, SQL view definitions enacting the communication were generated from ConceptBase models.

3.2 Reverse engineering of database schemas

In the previous applications, meta classes were used to define syntax and semantics of modeling languages (notations). The process of modeling is then the instantiation of the meta classes by appropriate classes.

Is that also possible when one has to reverse engineer conceptual models from logical designs? This has been investigated for the case of database schemas [17]. The idea of this application was to represent the source schema (e.g., relational schema) in the ConceptBase repository and then analyze it by suitable queries. The target schema (e.g. an ER diagram) should then be generated based on the analysis.

The approach proved to be valid, however only the analysis part was representable by the expressive means of ConceptBase. Source and target data modeling languages were modeled as meta classes. The relational data modeling languages includes features to declare primary keys and foreign key occurrences. Since the solution should also work for other data modeling languages, a method had to be developed to analyze input schemas independently from their data modeling language, i.e. their notation.

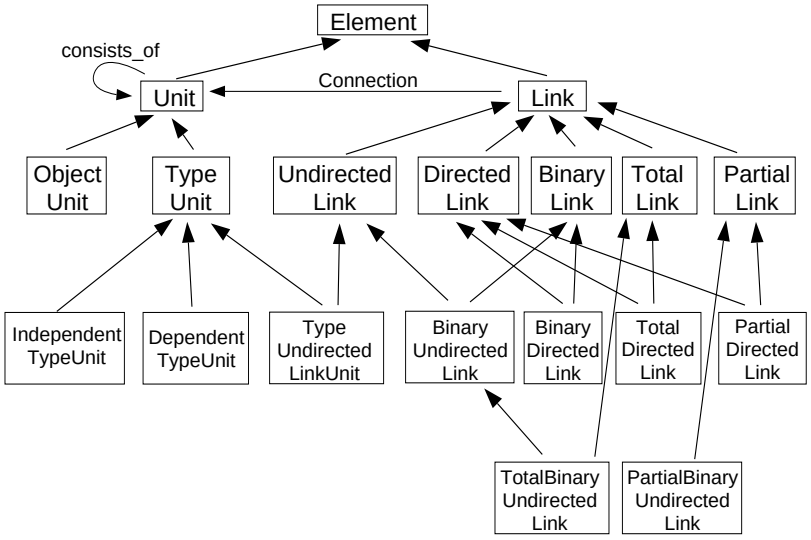

Fig. 2. Meta meta model to classify data concepts

Figure 2 shows a hierarchy of meta meta classes that are used for the analysis of input database schemas. It defines data element types to be either links or units. The concepts of the data modeling language are classified into this hierarchy. For example, *EntityType* is classified as *TypeUnit*; *Relation* is classified both as *TypeUnit* and *Link* because they play a double role. Now, database schemas can be described as instances of the suitable meta class. The difficult problem is now to classify concepts of the database schema into the hierarchy of meta meta classes. Note that the knowledge about the classification of data modeling concepts like *Relation* is far to imprecise. To find out whether a relation represents a *Link* one has to analyse the foreign key occurrences. This is done by reformulating connections of links in terms of foreign keys. In ConceptBase, this is done by deductive rules.

Given these definitions, one can then regard the concepts of the meta meta model as queries. For example, a binary link is a link which has exactly two connections to units. Since the property of having connections has been reformulated for the relational data model, one can use this query to find out those relations (in the source database schema) which are actually binary links. This applies to all meta meta classes of figure 2. Analysis of the source database schema is therefore done by evaluating the queries attached to the meta meta classes. For a given input concept, the most specific meta meta classes into which it is classified defines the analysis result.

The generation of the target schema (e.g. the corresponding ER diagram) cannot be done by query evaluation since it is about creating new objects. Moreover, target data modeling languages typically allow multiple ways of representing the same information. Thus, the generation was done by an external program.

Discussion. The analysis of database schemas can be implemented by queries which define the extension of abstract concepts. Since analysis for reverse engineering is about understanding the kind of a database schema concept, a hierarchy is a natural choice. Interestingly, the description of the hierarchy of meta meta classes is more complex than the description of the data modeling languages. The more meta meta classes are defined, the more precise is the result of analysis. The novelty of this application is the combination of a meta modeling approach with query evaluation: queries at the level of meta meta classes range over concepts at the schema level. The concepts of the notation level (meta classes defining the data modeling languages) appear as free variables in those queries. The project also showed that a declarative update facility would be advantageous. The implementation of the target schema by an external program is much less elegant than the analysis via queries. Active rules or constraint logic programs are potential candidates for update languages. As a whole, the meta modeling and the analysis features of ConceptBase were exploited. The application broadened the understanding of the wide usability of queries in information systems development.

3.3 A FUSION meta model in ConceptBase

Meta modeling in ConceptBase means to define a new (or existing) modeling language with the facilities of ConceptBase, i.e. meta classes and the logical sublanguage. A good example is FUSION, modeled in ConceptBase by Eckard von Hahn [8] in a student project at the Technical University of Munich. FUSION is an object-oriented software development method created by Hewlett-Packard. It distinguishes an analysis phase (description of classes, their relations, their operations, and constraints on allowable sequences of operations) from a design phase (mapping to run-time system objects, communications paths, inheritance hierarchies, method signatures). The analysis phase is characterized by frequent feedback to users. The information is captured in three interrelated models. The design phase is supposed to have no scheduled interaction with the user and consists of four models (interaction graphs, visibility graphs, inheritance graphs, and class descriptions). The last phase, implementation, is not specifically supported by FUSION models. The goal of the application project is, to represent all FUSION models (more exactly: modeling languages) in ConceptBase including their graphical layout and semantic constraints (of the modeling language). The class model of FUSION could be mapped rather straightforward to a ConceptBase meta class *ObjectClass* with attributes for roles and invariants. The second key meta class is *OperationSchema* with attributes to accessed objects, agents, and pre/postconditions. FUSION also includes a life cycle model which represents how (human) agents modify the system. A meta class *SystemState* describes the values of attributes attached to *SystemObjects*. The models of the design phase are left out in this brief overview. An interesting aspect is the representation of constraints. FUSION imposes basically two kinds of constraints on models. Syntactic constraints prohibit dangling references to undefined objects and enforce that any object has a unique name. So-called semantic constraints require that

values are instantiated from their types (e.g. *Integer* may only have integers as instances). Furthermore, cardinality constraints can be expressed and general integrity constraints. To give an example, the life cycle model of FUSION requires that each input event either triggers a call of a system object or the error handler. This is expressed by a simple static constraint as follows. Note that the constraint can only be expressed because the class *InputEvent* has attributes linking it to *SystemOperation* and *ErrorHandler*.

```
FusionMetaModel InputEvent with
  attribute
    callsA: SystemOperation;
    callsB: ErrorHandler
  constraint
    c1: forall ie/InputEvent
         (exists so/SystemOperation (ie callsA so)) or
         (exists eh/ErrorHandler (ie callsB eh) $
end
```

The application project also includes a case study from the petrol industry where the ConceptBase meta models of FUSION are used to represent the analysis and design models. These models are formally instantiated from the above-mentioned meta classes.

Discussion. The representation of FUSION in ConceptBase and the case study revealed some positive and some more critical remarks by the author. The (graphical) user interface was considered as easy to use and efficient. The predefined attribute categories for single and necessary attributes were appreciated[3]. The author argues however, that more general cardinality constraints like in the ER model should be included as well[4]. Performance was regarded as a major problem. The FUSION models contain quite a lot of constraints whose evaluation took several minutes, sometimes causing timeouts [5]. A more subtle point was the strict enforcement of constraints. A constraint is guaranteed to be true in all states of the ConceptBase database. If for example an attribute is defined as necessary, it must be present when an object is inserted to the system. This prohibits interactions where necessary components are incrementally inserted to ConceptBase. The error messages of the system were criticized due to their cryptic language. Incremental changes were difficult when a change caused several integrity violations.

The problem of propagating error information from the server into client views has meanwhile been partially addressed by methods for incremental maintenance of externally materialized views [25].

3.4 Requirements engineering for telecommunication services

FUSION is an example how a known modeling method can be realized by representing it as a meta model in ConceptBase. It may be argued that this is a pure academic exercise and that the meta modeling approach is more significant with the development of new modeling methods. The Requirements Assistant for Telecommunication Services (RATS) [4] validates that this is actually possible.

[3] Such attribute categories are used to attach properties to attributes of classes. For example, the ‘student-id’ attribute of a class ‘Student’ can be made instance of the ‘single’ attribute category defined in the meta class ‘Class’. An integrity constraint attached to it declares the semantics of ‘single’. Note that this integrity constraint will quantify over instances of ‘Class’ (like ‘Student’) and their instances. Such meta level formulas are essential for defining modeling languages inside ConceptBase.

[4] This is now possible to a degree by using the COUNT predicate offered in ConceptBase V5.0.

[5] In the meantime, two enhancements have been made to the query evaluator of ConceptBase. First, the execution ordering of predicates is optimized based on selectivity. Second, logical expressions are mapped to an algebra. Together, these two optimizations yield two orders of magnitude in efficiency.

RATS was developed as a prototype for British Telecom. The developers argue that there is a lack of CASE tools for this area, esp. for formal requirements capture. There have been methods developed for specifying such services, most prominently the Specification and Description Language (SDL) but it lacks support for requirements engineering. The RATS tool fills this gap and also provides for a mapping of requirements to SDL constructs. Special attention is devoted to validation (or verification) of the requirements models. As these are representations of potentially diverging user's views, this can only be accomplished by an approach which includes feedback to the originators of the diverging views.

The authors of RATS break the task into pieces by providing a collection of modeling languages. Like in FUSION, they are encoded as ConceptBase meta classes. Moreover, the authors propose to separate a brainstorming phase (ideas for requirements are generated by users without enforcing many restrictions) and a refinement phase (grouping of requirements into functional blocks for telecommunication services that can be mapped to SDL). Completeness is achieved by checking whether certain necessary components for service description are present (using so-called service definition templates). The modeling languages mentioned before are broken down as follows:

- Non-functional requirements (NFR) are quality goals to be achieved by the system (the telecommunication service). For example, a call must be acknowledged within 0.1 sec. NFR's are organized into a subclass hierarchy.
- Use cases are encoded sequences of system behavior and user-service interaction. While traditional approaches focus on textual, i.e. unstructured, representations, the RATS approach promotes a more structured method: a use case is given by its name, involved actors, a textual description, preconditions, flow conditions, and post conditions. In case of a single actor, a use case is described by sequences of atomic actions with input and output.
- Domain models are very diverse and large. They are decomposed into standards for telecommunication, expert knowledge, and temporary domain models relevant for the telecommunication service under development. Furthermore, the domain models include customer profiles (e.g. the subscribed networks of a customer), the customer's equipment (e.g. ISDN phone), characteristics of networks, interactions between features of networks, and finally a data dictionary which precisely defines meanings of (data) terms.

Each of these requirements has history attributes to store which stakeholder has specified the requirement at which time. The domain models are considered as a global knowledge base which exists before developing the functional and non-functional requirements of the new telecommunication service. Domain models and requirements models are linked by keywords and explicit attributes to concepts of the domain model. As domain models refer to existing telecommunication infrastructure this is a way to represent how requirements are going to be implemented.

The RATS tool relies on ConceptBase rules and constraints to guide the development process. The meta class representation already enforces certain constraints like referential integrity (all referenced concepts must be known). This is augmented by user-defined constraints of the different models. Some constraints are permanent (may never be violated). Others are so-called soft constraints. They are triggered only at certain milestones[6]. Active guidance is given to the developers by textual descriptions attached to classes (telling developers how to create instances) and by attributes which tell the developers how to continue (which concepts have to be defined next). Discussion between stakeholders is supported by 'agreement' attributes which encode the maturity level of some requirement. Constraints enforce that a requirement as a whole is agreed only if all its parts are agreed.

[6] The RATS tool does that by inserting these constraints temporarily at the milestones. If they are not violated, the development can go on. Technically, ConceptBase queries are perhaps more suitable for this task.

Discussion. The authors of the RATS tool discuss the trade-off between a "nice" model and performance. The more generic and reusable a model is (by modeling generic classes), the more instances can be expected for a class which potentially affects performance. Response time of around 30 secs by the ConceptBase system were considered the maximum. This prevented the authors from implementing too many classes and logical conditions though these would have been in principle desirable. The RATS tool is a comprehensive system for capturing requirements and mapping them to SDL. All requirements are stored in one repository (ConceptBase) and cross-related. The main difference to FUSION are the negotiation support and the domain models. This leads to a larger set of ConceptBase (meta) classes necessary for encoding the RATS methodology. Even more than with FUSION, performance was considered critical. It even constrained the extend to which the RATS method is encoded in ConceptBase. The expressiveness of the meta class mechanism and the logical sublanguage of ConceptBase were regarded as sufficient.

3.5 Design of attribute categories for modeling languages

Meta classes in the FUSION and RATS applications have features (attributes) that constrain the way how classes are built from these attributes. For example, a metaclass may provide an attribute called 'single' to which a constraint is associated about the single-valuedness of attributes. Then, a class 'Employee' which defines an attribute 'emp-id' as an instance of the metaclass attribute 'single' may only have instances with not more than one filler for 'emp-id'. The principle of attaching rules and constraints to meta classes is one of the major strength of ConceptBase because it allows to capture parts of the semantics of modeling notations (here: the cardinality of attributes). Dahchour [2] has applied this principle to investigate new abstraction principles for modeling notations.

The new abstraction principle is called 'materialization'. Intuitively, it is located between the specialization principle (subclasses of classes) and classification (instances of classes). The author presents an example of the automobile industry. A class 'CarModel' has attribute definitions for name of the car, number of doors etc. These attributes are tagged by tokens which indicate how the attributes are being used when building a car instance. A materialization class 'Car' of CarModel would add attributes for manufacturing date, serial number etc. Now, 'CarModel' can be instantiated, e.g. by the car model 'FiatRetro' which has 3 or 5 doors and an airbag, alarm, and cruise as fillers of the attribute special equipment. The materialization operator then creates a class definition 'FiatRetro_Cars' out of this where 'number of doors' becomes an attribute with possible fillers 3 and 5, and for each filler of special equipment a new attribute is created, i.e. one attribute for airbag, one for alarm, and one for cruise. These three attributes are string-valued. With these attributes, an actual instance like 'NicosFiat' can be constructed.

The author describes the materialization abstraction by ConceptBase meta classes and a collection of roughly one dozen rules and constraints. With these definitions ConceptBase can be used to test the effects of materialization and compare it to known abstractions like classification and specialization. The advantage of materialization is that it allows new ways of using class attributes: attributes of concrete classes (like 'Car') are instantiated in the ordinary way whereas attributes af abstract classes (like 'CarModel') can be expanded into different facets (like for the special equipment) before instantiating them.

Discussion. Whether materialization is a successful new abstraction principle for modeling notations has still to be seen. The case study shows however the strength of ConceptBase in designing new abstraction principles. The well-known principles specialization, classification, and attribution can be augmented or replaced by new principles without leaving the framework of deductive databases. Essentially, ConceptBase is used here as a validation platform for new modeling notations and abstraction principles. The rules and constraints of the new abstraction principles are the axioms of the new modeling notation. One has to be careful however: ConceptBase can answer queries and

check constraints against the database state. It cannot detect whether constraints are contradictory or subsumed by another. Dahchour found the expressiveness of the logical sublanguage sufficient except for the case of cardinality constraints. ConceptBase provides existential and universal quantification but it cannot easily count fillers for variables in a constraint formula. The latest release has a COUNT operator for the query language which could in principle be integrated into the constraint language.

3.6 Model analysis in cooperative design

The previous application concentrated on meta modeling with ConceptBase for defining new or existing modeling languages. ConceptBase is also strong in managing the models created by instantiating the meta models. It stores all objects in a main memory database subject to querying. If a model is reasonably small, it can be visualized and analysed by a human expert. Such an approach becomes intractable when models become very large, i.e. thousands or even hundreds of thousands of interrelated concepts. In 1994, the consulting company USU specialized in business process analysis contacted the ConceptBase team with this problem: analyze business process models for potential flaws. The details of a business process are gained from users in workshops and drawn as a series of diagrams (some for document exchange, some for task delegation, some for task execution of documents).

The solution to this problem [22] is to represent the notations for the diagrams by suitable meta classes (comparable in nature to the abovementioned approaches). Then, the diagrams are entered as instances of the meta classes into ConceptBase. Finally, queries are incorporated to find the potential flaws. The new aspect here is the emphasis on querying. There were more than 80 queries defined. Each query defines a pattern of a potential flaw. This is different from constraints defining the syntax and semantics of a modeling notation. A violation of a constraint is indicating an error that must be removed. However, certain patterns found in a (business process) model might indicate an error depending on the context.

The following example shows such a query for a system model (represented as a version of a data flow diagram). If an action updates a data element but not the complex data element which contains it, then this complex data element may be inconsistent. An example are computerized tax declarations. If the tax rate is changed by some action but not the tax amount (another data element of the tax declaration) then the whole tax declaration is inconsistent. If however only the address of the tax payer is changed then no further updates are needed. This shows that partial updates are in some cases harmful and in other cases they are not. The modeler has to decide based on external knowledge about the concepts of the model.

```
QueryClass partiallyUpdated isA Data with
  computed_attribute
     mainData: Data;
     updatingAction: Action
  constraint
     qc75: $ (~mainData containsPart  ~this) and
           (~updatingAction output ~this) and
           not (~updatingAction output ~mainData) $
end
```

In this application, queries encode modeling experience. The consulting company recalled from previous modeling projects the patterns that led to errors. The patterns are then coded as queries and added to the collection of pre-defined queries in ConceptBase. Since queries are just special forms of classes and classes are just objects, inserting or removing a query is a simple update. Additional

to the collection of queries, the consulting company created a user manual which exactly specified in which phase of the modeling project which queries should be submitted and how to react to the answer. The manual even contains estimates how long a certain query will take to evaluate. At the time of the case study, queries took up to ten minutes. Recent enhancements of the ConceptBase query optimizer have reduced answer times considerably, however.

Discussion. The above application emphasizes the role of model analysis. Traditional CASE tools emphasize the development of syntactically correct specifications which are ultimately mapped to program code. ConceptBase is a database system: by storing the models in a database they are subject to querying. As shown by the USU case, queries are more flexible than syntax checks. They can reveal shortcomings of models whose cause is outside of the model. The more the consulting company learns about which patterns can potentially lead to system flaws, the greater is their ability to avoid these flaws in the future. Thus, queries are an integral part of a modeling method. It is obvious that good performance of the query evaluator is essential for this modeling approach. The more queries can be evaluated in acceptable time, the more patterns of potential flaws can be described and detected. Another consequence of these experiences was the introduction of a module concept into ConceptBase, with explicit maintenance of knowledge about inconsistencies [21].

3.7 Quality management for data warehouses

Data warehouses are complex systems which pass massive data from online databases of an enterprise to analysis tools. The analysis tools support the decision making in the enterprise. Obviously, the quality of the decisions depend on the quality of available data wrt. precision, timeliness, completeness, consistency and others). The project DWQ investigates the foundations of quality management of data warehouses. ConceptBase serves as the prototype of a data warehouse meta database that supports quality management [11]. Data warehouses already incorporate meta databases as repositories for the list of available data sources and similar information and the idea was to extend them to cover quality management.

As in DAIDA, the ConceptBase system is used both for modeling quality management and for enacting it at run-time. Quality management is impossible without measuring components of the data warehouse. Thus, a model had to be developed which was capable to represent all major components (data sources, wrappers, views, control agents, etc.). Furthermore, the model had to cope with different perspectives on the data warehouse:

- the conceptual perspective contains conceptual models about data sources, the enterprise, and the clients (decision support tools); this includes conceptual data models like the ER model which clearly define subsumption between concepts,
- the logical perspective basically provides the data structures for the conceptual perspectives; for example, the relational schema of the data sources,
- the physical perspective defines the physical components like data stores, wrappers etc. together with their network addresses.

Essentially, this makes a design repository for a data warehouse system. Each of the objects in the repository is subject to measurement. For example, the completeness of a data warehouse schema (logical perspective) can be measured in terms of the number of conceptual enterprise objects covered by the schema. External measurement agents are used to compute quality values outside of the repository. More concrete, the completeness of a source relation, e.g., can be measured by counting the null values in the relation. Based on the quality measurements, a collection of queries is used to classify the quality of aspects of the data warehouse. These quality queries are then linked to quality goals of stakeholders.

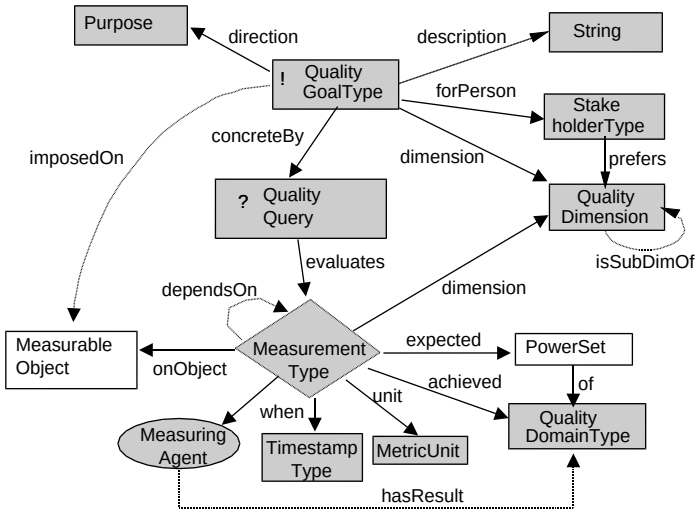

Fig. 3. Graphical representation of the quality meta model

Figure 3 gives an overview of the quality meta model. The gray parts are meta classes whereas the class 'MeasurableObject' is a meta meta class. This abstraction mismatch is not an error but a consequence of the nature of quality measurement: class-level objects like a relation are mapped to instance-level objects (a value for a measurement of this object). ConceptBase is suitable to model the crossing of abstraction levels since all information is stored in a uniform data structure. The quality meta model is derived from the Goal-Question-Metric approach of Basili and Weiss[1]. The coding as meta classes makes it to a part of the meta database of the data warehouse. Since the meta model abstracts from the notations for the various perspectives, it has to be instantiated in order to relate it to concepts like 'EntityType', 'Relation', and 'Wrapper'. The result is a collection of quality goals, quality queries, and quality measurements that are all stored in the meta database. Since they refer to notational concepts, a second instantiation has to be done to formulate quality goals, queries and measurements for a given data warehouse. The first instantiation yields a collection of reusable quality plans. This constitutes *knowledge* about quality management in data warehouses.

The enactment of quality management is done on the second instantiation. Measurement agents are storing their results as instances of quality measurement plans (first instantiation). Predefined quality queries are evaluated and yield reports about the current quality state of the data warehouse.

Discussion. DWQ yielded two important insights. First, a meta model needs not to be restricted to (meta) classes which are all at the same abstraction level. It is well possible that meta classes are related to meta meta classes. Second, the query language of ConceptBase is used to answer queries about properties of the environment. A collection of automated measurement agents is incorporated to gather the values of these properties. Since the values are of numeric nature, new operators to manipulate values in ConceptBase had to be implemented. This include simple arithmetic operators, aggregate functions, and advanced statistical functions. The latter have not been implemented so far. The project also created a demand to manage analytical models about dependencies between quality values. The idea may be transferred to software metrics in CASE repositories as well: the quality of software objects can be assessed based on a quality model coded in the repository. The manager of a software project can then evaluate the state of the project by querying the repository. Changes to

quality management policies are implemented by making appropriate updates to the quality model in the repository.

3.8 Managing hypertext books

A classical type of meta data are schema descriptions of information systems. Web-based information systems are often simply collections of hypertext documents and are only weakly structured. Various systems were developed during the last years for providing some form of integrated view and uniform accessibility on such data (cf. literature on semi-structured data). Website management systems mainly follow a model-based approach where the web page contents is generated from a database, considering the schema as well as data. In both cases the (virtual) schema and even additional access and integration can be managed in repositories like ConceptBase. As a special application the Web enabled also new types of books. The conventional way of reading books (usually in a sequential way, sometimes based on an index or table of contents) completely changes wrt. the media as well as the navigational facilities. So-called hyperbooks are groupings of electronic documents which are understood as an entity and have a *model-based* description of their structure and contents.

The KBS Virtual Classroom Project launched in 1996 at the University of Hannover belongs to one of the initiatives in Germany for exploiting web technology and multimedia for teleteaching. One outcome of the project is the KBS Hyperbook system [5, 20] for managing and presenting hyperbooks, practically applied mainly for computer science courses. In the system architecture ConceptBase serves as central storage component for the model describing the book contents as well as for the plain elementary page references which are used to dynamically build the compound pages presented to the reader. The KBS hyperbook model is based on three layers of abstraction:

1. The hyperbook 'basic abstraction' comprises the general notion of *hyperbookObject* as root abstraction and certain main object types such as *Concept, Relation, WWWPage*. Relations are distinguished wrt. their cardinality and special categories *Inheritance InstanceOfRelation*.
2. The 'structural hyperbook abstraction' provides two types of basic information carriers, namely *SemanticInformationUnits* and *ProjectUnits*. The latter serve for illustrating the usage of the earlier in specific contexts and are linked together by *knowledge items*. Additional components, e.g. *trails* and *goals* stand for possible paths through the concepts and represent certain intentions of users resp.
3. The 'semantic modeling abstractions' consist of the domain specific concept hierarchies relevant for a concrete hyperbook including subhierarchies typically overlap and provide occurrences of concepts in various contexts.

These abstraction layers in part have some form of 'refinement' relationship from the semantic up to the basic abstraction, but they also respect additional aspects and contribute to one of the four *model views*: The *domain model* contains both the semantical abstraction layer concepts including links to the representation as web page fragments and concrete example objects, which instantiate these concepts and are used mainly for demonstration purposes (e.g. a sample Java program). The *navigation model* shares parts of the structural abstraction with the *user model*. Finally, the *visualization model* provides elements for specifying page fragments e.g. by MIME objects.

Despite of the different abstraction layers the implementation with ConceptBase does not reflect a corresponding IRDS like multi-level presentation. The actual hyperbook meta model of the implementation consists of a combined model of all perspectives mentioned above apart from the domain model, which actually is the data describing a concrete hyperbook. Nevertheless, we have additional instantiation relationships on the data level, e.g. between instances of *SemanticInformationUnit* where a specific one instantiates more abstract properties of a generic one. In this context

[20] emphasizes the importance of possible multiclass membership in Telos, in particular that this is allowed to occur spanning different model layers. As an alternative to the design finally chosen the authors discuss e.g. that it should be possible to shift down even the very abstract notion of *SemanticInformationUnit* from the meta to the data level such that this concept itself can be described in the hyperbook (i.e. by page components explaining its meaning).

The standard web viewer based GUI communicates with servlets on the KBS Hyperbook server. Depending on the user specified concept to be explained or trail to be followed, the servlets collect all necessary information from ConceptBase including links to external Web page fragments separately stored inside the server file system. Whether links should exist between the various fragments is also derived by ConceptBase through a set of deductive rules belonging to the navigational part of the meta model. The rules either take relationships between objects in the domain model (e.g. pages describing sample objects can be reached from the pages of its classes) into account as well as trails based on so called trail indexes. A second application of deductive rules concerns the dynamic visibility of links resp. Web page fragments. This aspect is used as part of the user model but on the data level by specifying, e.g., which pages a user must have read before he can continue to fetch another document.

The queries issued to ConceptBase in order to construct the output pages are based on the domain model and the navigational model and in addition respect the user model. More precisely they retrieve all relevant representation fragments to a certain concept and the references to other concepts focusing on the respective user view or a current trail. Both queries and rules resp. integrity constraints were identified to be a main advantage of Telos over other modeling languages like OMT because they allow the definition of new kinds of constructs and relationships. One design decision for the KBS hyperbook implementation concerns the explicit introduction of a query cache between servlets and ConceptBase in order to avoid unnecessary communication and repeated computation of the same results. This is a hint for certain performance problems or querying on a too detailed level.

Discussion Although the requirements to hyperbook modeling obviously differ from the previous meta modeling approaches, ConceptBase was able to support even this. The meta level in this case simply consists of the hyperbook generic model parts whereas the data level contains the hyperbook specific domain concepts, parts of the navigational information and representation aspects. In this respect, the implemented Telos version in ConceptBase differs from the original definition by not axiomatizing the membership of stored objects in classes, metaclasses, etc. The levels *can* be used to reach a more systematic understanding but it is not obligatory to structure the model in such a way. The deductive rule language could be used to avoid the laborious manual specification of links by implicitly specifying rules on the meta level. The query language covered the necessary aspects for interacting with the Java servlets used to build up the Web page structures. Probably the extended view language in the current version of ConceptBase may be helpful to specify more complex building blocks of the pages to be presented to the user. The KBS application utilizes the Internet connectivity of ConceptBase (to generate hypertexts out of models stored in ConceptBase). Moreover, the meta modeling capability is used to represent domain concepts as well as features for external representation[7]. The analysis capability of ConceptBase plays a minor role in KBS due to the navigational style of user interaction with hypertexts.

[7] The KBS hyperbook approach has been applied to design the online textbook of a computer science course. There, students learn generic concepts as well as examples of them, e.g. *ProgrammingLanguage* vs. *Java*. Since both concepts have to be treated uniformly, the KBS meta model allows concepts to be instances of more than one (meta) class level.

4 Historical Remarks and Development Strategy

We started development of ConceptBase in 1987 at the University of Passau. Version 1.0 was available in early 1988. Thus, ConceptBase was one of the first deductive object managers. Version 2.0 appeared in November 1989 and added deductive integrity checking plus the client server architecture. A few months later an interval-based time calculus was included. Version 3.0 was released in 1991 to provide query classes for information access and a X11-based user interface. In 1992, the ConceptBase development team moved from Passau to the RWTH Technical University of Aachen. Two intermediate versions 3.1 and 3.2 were issued subsequently which basically improved performance and reliability. Version 3.2 had outstanding robustness. This was due in part to the fact that the interval-based time was removed from the system. Subsequent ConceptBase versions just provide rollback functionality based on transaction time. Version 4.0 was finished in 1994 and then upgraded to 4.1 after one year. It provided a dedicated object store and boosted performance. For the first time, real meta class-level rules and constraints were made available. The current version is 5.0 and features complex views, modules, active rules, and a much improved query optimizer.

The development strategy was neither purely user-driven nor purely technology-driven. It has been a mixture of both. Before a development cycle starts, a review of user feedback and a study of current trends in the application and technology domains is made to select candidate functionalities to be included in the next release. Depending on the available resources and research interests, the new functionalities are implemented and tested internally. In some cases, the actual benefit is unsatisfactory and the new function is excluded from the next release. For example, a group security mechanism was developed in the early 90'ties but its overhead was too high to be acceptable. Still, the research results were very useful. They are now reconsidered for implementation after the base performance of the system has dramatically improved. Over the years, the emphasis of development moved from the kernel (object store, logic formula evaluation) to the application side (user and programming interface, web-ification, access to heterogeneous data sources). Extensions to the kernel are still made from time to time. For example, new predicates were provided for better access to instance level attribute labels because an application had an urgent need for them. In order to improve the feedback from ConceptBase users to the development team, the *ConceptBase Forum* was established in 1998. Members of this Web-based workspace can upload and download background material, formulate technical questions and answers, and access a library of modeling examples for different purposes.

Besides the positive application reported in the previous sections, there have also been unsuccessful projects with ConceptBase. A frequent cause of failure is the use of ConceptBase as a plain object store where complex objects are retrvied by a given object identifier. While ConceptBase supports this operation, the overhead in decomposing a complex object into its parts at the application program side is generally too high. Another potential failure cause is the extensive use of integrity constraints. Applications programs are usually not prepared to react intelligently to messages reporting a violation of an integrity constraint. Even if the interaction is directly with the user, it can be a non-trivial effort to define an update that does not violate any integrity constraint.

5 Conclusions

The specific demands of the application projects has guided to development of the ConceptBase system over the last ten years. It is fair to say that ConceptBase represents the lessons learned from quite diverse aspects of information systems developments (ranging from design repository to hypertext information systems) of various domains. ConceptBase can be classified as a Meta CASE tool, i.e. a tool whose system engineering capabilities are described via its meta modeling features. The process

to define such meta models is called method engineering. Since the models are represented as (meta) classes, they can be directly used as an environment for systems development.

The combination of the method and system engineering capabilities was used by most application projects extensivly. Therefore, the type of (meta) meta models developed in those application projects give some clue about the requirements for system development tools and for method engineering environments. If the application is about classify existing information, then the meta model tends to be a hierarchy (see reverse engineering application). In contrast to that, applications which have to manage inter-related modeling languages tend to have a few generic concepts linked together by attributes at the meta meta class level. This feature allows to formulate analysis queries that cross the boundaries of a single modeling language (see DAIDA and USU applications). Thirdly, if the application focuses on a single modeling language, then the meta model has similar size and structure to the syntax specification of the modeling language (FUSION).

We summarize the application experience with ConceptBase as follows.

1. *Performance determines the use of features.* This statement appears trivial but it is important when it affects method engineering. If certain features of a modeling notation cannot be implemented efficiently, then method engineers tend to leave them out of the method. Note that system development methods are described by ConceptBase meta classes with rules, constraints and queries. Hence, a superb query optimizer is needed to give method engineers the capability to cover the important aspects of a systems development method (like FUSION or RATS).
2. *Models can grow large and require tools for analysis.* The business process analysis application is a good example for the need to automatically analyze large models. A graphical visualization of a model (e.g. an ER diagram) is only suitable for relatively small models. Human developers tend to overlook (semantic) errors in a model especially when it is interrelated to other system models. ConceptBase represents all models in its database. This makes the models subject to extensive querying. The expressiveness of the query language was constantly improved in order to be able to make sophisticated analysis. There is certainly a tradeoff between computational expressiveness and efficiency. The design decision in ConceptBase was allow recursive queries but not to leave the logical framework of deductive rules.
3. *Meta modeling is a powerful facility for method engineering.* ConceptBase employs a rather simple principle for meta modeling: classes are objects and can be instance of other classes called meta classes. A system development method (like FUSION, RATS,...) is described by meta classes. The properties of the meta classes (rules, constraints, graphical symbols) are all attributes of the meta classes. Since meta classes are also objects they can be instances of other classes called meta meta classes. The layer of meta meta classes can be used to model the goals of a collection of system modeling languages. For example, in the case of database schema reverse engineering, the meta meta classes allow to represent source and target data modeling languages and to classify input database schemas into them.
4. *Multi-user applications require communication support.* System development is a multi-user activity. Communications acts refine the model or extract knowledge from it. A repository system like ConceptBase must have excellent facilities for supporting system and method engineers. The first prototype of ConceptBase was a single user system. Then, the database server was separating from the user interface. Communication between the user interface is done via two methods: ASK for extracting information and TELL for refining information. Recently, the query language was augmented by a view mechanism that allows to model how answers to queries are passed to ConceptBase clients. The answer materialized in the client can be kept up-to-date by the active rule and view maintenance capabilities of the ConceptBase server. It has been demonstrated in a recent diploma thesis that this is sufficient for implementing standard (as well as non-standard) groupware communication protocols.

It should be mentioned that the meta modeling feature requires much expertise. Meta modeling is a kind of method engineering similar to the design of a programming language. If a meta model is ill-designed, then the consequences are potentially tremendous since all class definitions using the meta model may have to be re-written. Therefore, a meta model should undergo a critical review and experimentation before being used on a larger scale for modeling. Up to now, we have not developed a methodology on how to do method engineering with ConceptBase. One building block is the multi-perspective approach [21] though it is specialized for systems engineering. Currently, the best way is to experiment with given examples and to adapt them to the application.

ConceptBase is still under continuous development. The communication support is being extended by wrapper agents that link to legacy information systems. The description of modeling languages purely by meta classes is certainly not comprehensive. Pragmatics of modeling methods are missing and a good way to communicate experience to modelers. Performance remains an issue esp. when the system is used to handle large amounts of information. Active rules have only been used by few ConceptBase application projects. An experimental conference management system utilizing active rules show their great potential in coordinating distributed teams. Much of the functionality of a workflow management system can be implemented with this feature. Another promising capability of ConceptBase are modules. Import and export interface specify which parts of the database are visible to a certain application.

An open issue is the provision of generic user interface tools. The meta-modeling features of ConceptBase make it relatively easy to specify customized modeling languages. However, this is not matched by equally adaptable tools of the ConceptBase user interface. The graphical browser does provide some options to control the graphical layout of nodes and links but this is not sufficient when compared to commercial CASE tools.

More information about the system is available on the Web site

`http://www-i5.informatik.rwth-aachen.de/CBdoc`

It includes instructions on downloading the system for research purposes. The site also offers a comprehensive list of literature about the system and its applications.

Acknowledgements. Many thanks go to the ConceptBase development team for their endurance over the last twelve years, esp. Rainer Gallersdörfer, Hans Nissen, René Soiron, Kai von Thadden to mention only a few. Funding for this work come from the ESPRIT program through projects DAIDA, COMPULOG, DWQ, MEMO, and the Deutsche Forschungsgemeinschaft in its Federal Research Program 'Objektbanken für Experten', and the German Ministry of Research through the cooperation projects WibQuS and FoQuS.

References

1. Basili,V.R., Weiss, D.M.: A method for collecting valid software engineering data. *IEEE Trans. on Software Engineering* **10**(6) (1984) 728-738.
2. Dahchour, M.: Formalizing materialization using a metaclass approach. Proc. 10th Intl. Conf. Advanced Information Systems Engineering (CAiSE'98), Spinger-Verlag, LNCS 1413, 1998.
3. Dömges, R., Pohl, K.: Adapting traceability environments to project-specific needs. *Communications of the ACM* **41**(12) (1998) 54–62.
4. Eberlein A., Halsall F.: Telecommunications service development: a design methodology and its intelligent support. *Journal of Engineering Applications of Artificial Intelligence* **10**(6) (1997) 647–663.
5. Fröhlich, P., Nejdl, W.: A database-oriented approach to the design of educational hyperbooks. Proc. Workshop Intelligent Educational Systems on the World Wide Web, 8th World Conference of the AIED Society, Kobe, Japan, 18-22 August 1997.

6. Greenspan, S., Mylopoulos, J., Borgida, A.: On formal requirements modeling languages - RML revisited. Proc. 16th International Conference on Software Engineering (1994) 135-148.
7. Greenspan, S.: *A knowledge representation approach to requirements definition*. Ph.d. thesis, Department of Computer Science, University of Toronto, 1984.
8. von Hahn, E.: *Meta-modeling in ConceptBase - demonstrated on FUSION*. Semester thesis, Faculty of CS, Section IV, Technical University of München, Germany, October 1996.
9. Jarke, M.: *Database application engineering with DAIDA*. Springer-Verlag, 1993.
10. Jarke, M., Gallersdörfer, R., Jeusfeld, M.A., Staudt, M., Eherer, S.: ConceptBase - a deductive object base for meta data management. *Journal of Intelligent Information Systems* **4**(2) (1995) 167–192
11. Jarke, M., Jeusfeld, M.A., Quix, C., Vassiliadis, P.: Architecture and quality in data warehouses - an extended repository approach. In Pernici, B., Thanos, C. (eds.): Special Issue on Advanced Information Systems Engineering, *Information Systems* **24**(3) (1999) 229–253.
12. Jarke, M., Jeusfeld, M.A., Rose, T.: A software process data model for knowledge engineering in information systems. *Information Systems* **15**(1) (1990) 85–116.
13. Jarke, M., Peters, P., Szczurko, P., Jeusfeld, M.A.: Model-driven planning and design of cooperative information systems. In M. Papazoglou, G. Schlageter (eds.): *Cooperative Information Systems - Trends and Directions*, Academic Press, 1998.
14. Jarke, M., Rose, T.: Managing knowledge about information system evolution. Proc. SIGMOD Intl. Conf. on Management of Data (1988) 303–311
15. Jeusfeld, M., Jarke, M.: From relational to object-oriented integrity simplification. Proc. 2nd Intl. Conf. on Deductive and Object-Oriented Databases (DOOD'91), Springer Verlag, LNCS 566 (1991) 283–295.
16. Jeusfeld, M.A., Jarke, M., Nissen, H.W., Staudt, M.: ConceptBase - managing conceptual models about information systems. In P. Bernus, K. Mertins, G. Schmidt (eds.): *Handbook on Architectures of Information Systems*, Springer-Verlag, 1998.
17. Jeusfeld, M.A, Johnen, U.A.: An executable meta model for re-engineering of database schemas. *Intl. Journal Cooperative Information Systems* **4**(2&3) (1995) 237–258.
18. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos – a language for representing knowledge about information systems. *ACM Trans. Information Systems* **8**(4) (1990) 325–362.
19. Mylopoulos, J., Bernstein, P.A., Wong, H.K.T.: A language facility for designing data-intensive applications. *ACM Trans. on Database Systems* **5**(2) (1980) 185–207.
20. Nejdl, W., Wolpers, M. KBS hyperbook - a data-driven information system on the web. Technical Report, KBS Institute, University of Hannover, Germany, November 1998.
21. Nissen, H.W., Jarke, M: Repository support for multi-perspective requirements engineering. In Lyytinen, K., Welke, R.J (eds.): Special Issue on Metamodeling and Method Engineering, *Information Systems* **24**(2) (1999).
22. Nissen, H.W., Jeusfeld, M.A., Jarke, M., Zemanek, G.V., Huber, H.: Managing multiple requirements perspectives with metamodels. *IEEE Software* **13**(2) (1996) 37–48.
23. Pohl, K.: *Process centered requirements engineering*. John Wiley & Sons Ltd, UK (1996).
24. Ramesh, B., Dhar, V.: Supporting systems development by capturing deliberations during requirements engineering. *IEEE Trans. on Software Engineering* **18**(6) (1992) 498–510.
25. Staudt, M., Jarke, M.: Incremental maintenance of externally materialized views. Proc. 22nd Intl. Conf. on Very Large Data Bases (VLDB'96), (1996) 75–86.