Metamodelling with Datalog and Classes: ConceptBase at the Age of 21

Matthias Jarke, Manfred A. Jeusfeld, H.W. Nissen, C. Quix, M. Staudt

RWTH Aachen University & Fraunhofer FIT, Ahornstr. 55, 52074 Aachen, Germany
Tilburg University, The Netherlands
Cologne University of Applied Sciences, Germany
Munich University of Applied Sciences, Germany
jarke@cs.rwth-aachen.de

Abstract. ConceptBase is a deductive object-oriented database system intended for the management of metadata. A distinguishing feature of the Telos language underlying ConceptBase is the ability to manage rules and constraints across multiple levels of instantiation in so-called meta formulas, thus offering uniform consistency management across heterogeneous notations or ontologies. Originally developed in the context of model-driven database design in the late 1980's, ConceptBase has been used in several thousand installations all over the world for numerous applications in areas such as requirements engineering, engineering information management, model management, eLearning, cultural information systems, and data warehousing. The internal representation is based on a quadruple object structure, combined with advanced Datalog engines, such that many optimization techniques in ConceptBase have pioneered ideas later pursued in the implementation of XML databases and ontology-based reasoning and data management engines.

1 Introduction

The large number of different modeling formalisms used in information systems engineering, semi-automated development techniques such as Model-Driven Design, but also the increasing richness of media handled by such systems beyond the traditional structured data, has renewed the interest in so-called metadata repositories and model management systems since at least the end-1990's. In standards such as the Information Resource Dictionary Standard IRDS [11] or OMG's meta object facility MOF [31], but also in many experimental and commercial systems such as, e.g., MetaEdit+ [20], Clio [10] or Rondo [24]. A shared feature of these standards is that not just data and their schema or other metadata are stored but also the metaschemas for these metadata and their relationships. In the typical heterogeneous environments, further metalevels may be necessary to manage the relationships between different metaschemas or modeling languages, such that a multi-level hierarchy of instance-class relationships ensues. It is surprising to see that, despite this obvious and

increasing need, after more than 20 years, our ConceptBase system is apparently still the only one that offers full support for the syntax and semantics of such multi-level hierarchies with heterogeneity at all levels. This paper reviews some of the features of ConceptBase that made this possible as well as some of the many applications in research, teaching, and practice the system has enjoyed and continues to enjoy.

The development of ConceptBase was motivated by work in the European DAIDA project [15], in which an early version of what would now be called model-driven information systems development was developed, using a mapping from semi-formal requirements modeling languages [8] via the design language Taxis [25] to database programming languages. A repository was needed to document and maintain the developed artefacts as well as their relationships from a product, process, and design tool perspective, ensuring traceability and incremental design within and across multiple design versions. A version of the Telos information systems modeling language [26] formed the formal starting point for the ConceptBase development but the final version of Telos itself was also heavily influenced by the application domain of metadata repository management.

We started the development of ConceptBase in mid-1987, the first version became operational in late 1988 [16]. About a year later, a stable client-server version existed which was already used in 1989 as perhaps the first Internet-based knowledge base management system in a project on requirements traceability modeling across the Atlantic ocean, four years before the advent of the World Wide Web. Using innovative storage models similar to the ones nowadays used in XML stores, and extending optimization techniques for query and integrity processing in deductive databases for our case of a deductive object-oriented metadata manager, the performance of ConceptBase improved rapidly, leading to a stable and externally usable prototype by about 1993 [12]. In the rest of the 1990's, about 250 applications in various domains of research, teaching, and even industrial practice became known to us, some of them with our participation but many also completely independently.

In the new century, the user community of ConceptBase increased further, probably due to the broadened interest in model management and metadata management for multimedia data where our experiments were also brought into some of the multimedia metadata standardization committees. Dissemination was also helped by new system features and many further performance improvements, plus a robustness of the system that is now competitive with many commercial systems. At present, we know of over 1000 registered installations, probably a number of unregistered ones exist as well.

In [18], detailed descriptions of the meta modeling context, the ConceptBase systems, and some of the more influential applications are described, and a current version of the system is made available with many examples. In this short overview paper, we first review the most important language features of ConceptBase and then give an overview of the application domains and the impact experiments with ConceptBase have achieved in these domains. We end with some indications of ongoing work.

2 Language Features of ConceptBase

ConceptBase is an implementation of the object model O-Telos [17], a Datalog-based variant of Telos [26]; for simplicity, we use the name Telos in the sequel. We first give a general introduction to the Telos language features in general and then highlight some features that distinguish ConceptBase from similar systems or have been added to the system rather recently.

2.1 Basic Concepts

As in all deductive database models, Telos databases consist of an explicit and an implicit part. All explicit information is reified in the form of objects with object identity. This holds for regular objects (instance level), for classes, meta classes etc., but also for non-derived instantiation, specialization and attribution links. For example, any explicit attribute is also an object and can have attributes itself.

To make this very general object concept possible, the basic object is a proposition `P(o,x,l,y)` in a kind of semantic network link labeled `l` with `o` is an object identifier (oid), `x` as the source oid and `y` as a target oid. Such a proposition has the dual role as a fact in the sense of deductive databases, and as an identifiable object in an object-oriented database, thus forming the elementary bridge within the deductive object-oriented approach of ConceptBase. It also allows ConceptBase to offer a textual syntax as well as an equivalent graphical syntax to the user. Both of them hide the object identifiers to the user and only work with the labels.

Note that this approach can be seen as a precursor to the very similar triple storage approach for XML or RDF [27], except that those do not work with object identifiers and thus offer a bit less flexibility. Among other things, this similarity implies that many of the storage and query optimization techniques developed for ConceptBase over the years can be evaluated for their applicability to semi-structured databases.

As special subkinds of propositions, Telos supports instantiation (instances, classes, meta classes, meta meta classes, etc.), specialization, and attribution. In the graphical syntax supported by the ConceptBase graph editor, these three kinds of links are typically indicated by graphical symbols, as shown, for example, in fig. 1. As shown in sec. 2.2, the user can extend the collection of such subkinds by meta-objects which are given semantics through meta-formulas.

Deductive rules and integrity constraints can be defined for objects at any abstraction level. In the textual syntax of deduction rules and constraints (see sec. 2.2 for examples), standard labels are used in literals for instantiation (`x in C`) and specialization (`C1 isA C2`) whereas, due to the greater variability of attribution link types, we offer the form (`x m/l y`) where `x` is the label of the source object, `y` is the label of the target object, `l` is the label of the link itself, and `m` is the label of the class of links to which the link belongs (also called the attribute category).

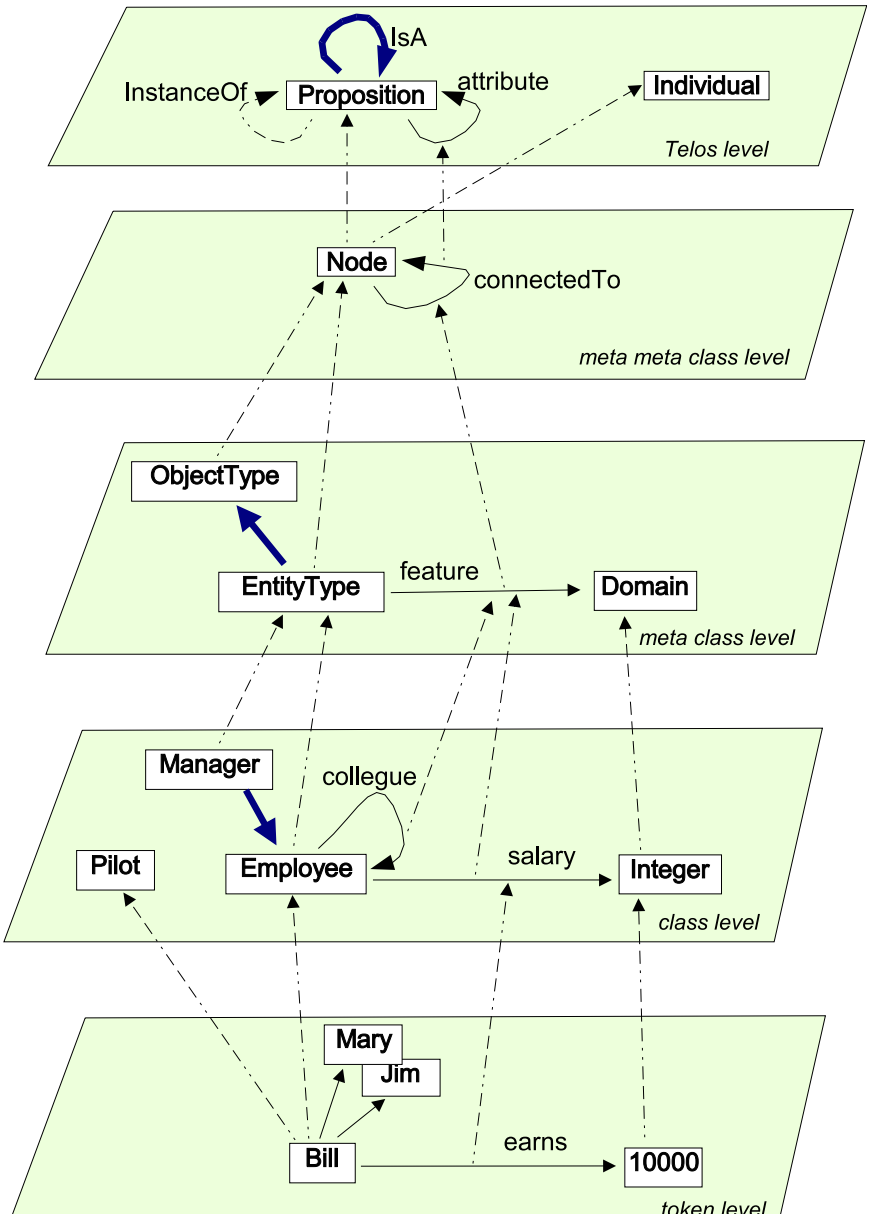

Fig. 1. A simple Telos knowledge base graph in five meta-levels: dashed lines indicate instantiation, bold lines isA, normal lines attribution; both nodes and links are objects in Telos

The external textual syntax of Telos is a frame syntax that groups a large number of propositions into a coherent and more easily understandable frame. It defines, for a given object, its classes (multiple instantiation is possible), its generalizations (not allowed for instances), and its attributes grouped by attribute categories. As a simple example, some of the objects in fig. 1 can be described in frame syntax as follows:

```
Bill in Employee, Pilot with
  salary
    earns : 10000
  colleague
    col1 : Mary
    col2 : Jim
end

Employee in EntityType with
  feature
    salary : Integer;
    colleague : employee

Manager in EntityType isA Employee end
```

A collection of basic axioms comprising a number of facts, rules, and constraints defines the semantics of Telos used in ConceptBase. For example, the `isA` relationship between `Manager` and `Employee` means that instances of `Manager` can also have the `features` `salary` and `colleague` by inheritance. We refer to [JJM09, ch. 3] for more details.

The example also illustrates the perhaps most distinguishing feature of Concept-Base which will be elaborated in more detail in the next subsection: an in principle infinite hierarchy of instantiation relationships allows ConceptBase as a repository system of design knowledge to manage a complete hierarchy of example objects/scenarios, their classes, their meta classes, their meta meta classes, etc. in a uniform framework governed by a well-defined syntax of a deductive database.

Of course, such a uniform framework is only useful if it can be efficiently processed. This was achieved by the important result in Manfred Jeusfeld's thesis [17] that the collection of Telos axioms for Telos enforces a deductive database which can be mapped to Datalog with dynamically stratified negation, and thus processed with high efficiency by any good Datalog engine. Indeed, while the early ConceptBase versions were implemented on top of Prolog systems linked to data stores, recent versions since 2002 rely on dedicated Datalog engines, thus leading to very competitive performance and to by now commercial-level stability of the whole system with large data sets. For readers interested in deductive databases, it should be pointed out that achieving the dynamic stratification was by no means easy given the fact that we have essentially only one single stored relation (of Propositions) in ConceptBase. One of the key solution ideas was to replace the generic `in` relationship for instantiation by specialized `in.C` relationships to class `C` using partial evaluation – the same trick we shall use in sec. 2.2 below for handling meta formulas efficiently.

In two further doctoral theses, supported by several master theses (too many to be mentioned here in detail), important practice-oriented extensions of the basic syntax and semantics of the language were achieved. Hans Nissen demonstrated that it is possible with just a few additional axioms and limited implementation effort, to add a module concept to ConceptBase. It allows, among other things, the team development and delayed consistency checking of large complex models [28]. With the concept of Query Classes, Martin Staudt invented a very flexible view mechanism which – like in SQL or (long after ConceptBase) XQuery -- to ensure closure in deductive repositories by making the results of queries ConceptBase objects; implementation of these objects nevertheless can adapt and extend all the ideas for efficient deductive query optimization, view maintenance, and integrity checking from the literature [34, 35]. An interesting application was our idea of externally materialized views in which a query class is materialized outside control of the system itself but incrementally informed about necessary changes to the view. Such algorithms could, e.g., be used to maintain materialized views on mobile devices with uncertain linkage to their data sources. We shall give examples of query classes below.

2.2 Meta-Formulas

In this subsection, we elaborate more how we accomplished the most distinguishing feature of ConceptBase – its handling of multiple instantiation levels as a pre-requisite for many model management applications in heterogeneous systems.

A deductive rule or integrity constraint typically ranges over exactly one abstraction level, i.e. it is defined at a certain level (e.g. the class level) and the variables range over objects at the next lower level (e.g. instance level). For example, a class `Employee` can have an attribute `salary` and a constraint that demands that the salary of an employee must be smaller than the salary of the `Manager` of his department. Another example is an integrity constraint that demands that instance of `EntityType` must have at least one attribute. Here, `EntityType` is a meta class and its instances are classes.

Meta-level formulas are formulas that range over objects from more than one abstraction level. For example, the key constraint in the relational data model is a formula expressed at the meta class level (the concept Relation is a meta class) but is evaluated against the database instance (instance level). The class level (database schema) is referred to by variables. Meta-level formulas are particularly useful for meta modeling, i.e. the specification of constructs of modeling languages.

As an illustrative example, consider a very simple process language, in which tasks have successor tasks. A task with more than one successor task is a 'predicate task' (condition). A task without successor is an end statement. A task that is not the successor of another task is a start statement. All other tasks are procedural tasks. Besides, tasks are executed by agents. We demand that there is a unique start statement and a unique end statement. We are interested in detecting loops. Moreover, we want to check whether there are agents who are executing two tasks t1 and t2, where t2 indirectly follows t1 but there is at least one task in between that is executed by another agent (execution split).

The structural part of this simple process language is defined in the Telos frame syntax as follows:

```
Task with
  attribute
     successor: Task
end
Agent with
   attribute
     executes: Task
end
```

To deal with the integrity constraints and the analysis queries, we need to be able to follow the successor link transitively. Since transitivity is frequently used, we specify it as a general construct with a meta-level formula:

```
Proposition in Class with
  attribute
    transitive: Proposition
  rule
    trans_R: $ forall x,y,z,R/VAR
                AC/Proposition!transitive C/Proposition
                  P(AC,C,R,C) and (x in C) and (y in C)
                   and (z in C) and
                   A(x,R,y) and A(y,R,z) ==> A(x,R,z) $
end
```

Note that the relation `R` is a variable in the formula. It is the label of any attribute `AC` that is required to be transitive. In our class definition of `Task`, we now simply make the successor attribute transitive via

```
Task with
  attribute,transitive
     successor: Task
end
```

Some subclasses of `Task` do not require transitivity, e.g.

```
StartStatement in QueryClass isA Task with
  constraint
    c1: $ not exists link/Task!successor To(link,this)$
end

PredicateTask in QueryClass isA Task with
  constraint
    c1: $ exists s1,s2/Task A_e(this,successor,s1) and
          A_e(this,successor,s2) and (s1 \= s2) $
end
```

In a similar way, we can define end statements and join statements (more than one direct predecessor). The predicate `A_e(x,successor,y)` operates on explicit successor facts, whereas `A(x,successor,y)` also operates on facts derived via the transitivity rule

```
LoopTask in GenericQueryClass isA Task with
  parameter
    rep: Task
  constraint
    c: $ A(this,successor, rep) and
         A(rep,successor,this) and
         (exists s/Task  A_e(rep,successor,s) and
                         A(s,successor,rep)) $
end
```

Hence, a task like `this` is a loop task for the loop represented by 'rep' if rep can be transitively reached from 'this' and rep can be reached from itself via at least one intermediate task s.

The execution split query is also exploiting the transitivity:

```
AgentWithSplitResponsibility in QueryClass isA Agent
with
   constraint
     c1: $ exists t1,t2,t/Task A(this,executes,t1) and
          A(this,executes,t2) and A(t1,successor,t) and
          A(t,successor,t2) and not A(this,executes,t) $
end
```

Figure 2 shows a graphical representation of the analysis of an example workflow defined by the query classes above. The queries are displayed as ovals. The answer to a query is the set of instances that fulfill the membership constraint of the query class. This derived instantiation is denoted by dotted links. Thus, `InsuranceAgent` is a derived instance of `AgentWithSplit-Responsibility`. A loop is detected as well featuring four loop tasks. The loop tasks `checkPolicy` and `proposePayment` are additionally classified as predicate tasks.

Note that the above query class definitions are sufficient to provide this functionality. Just by storing them in ConceptBase you get the desired analysis capability.

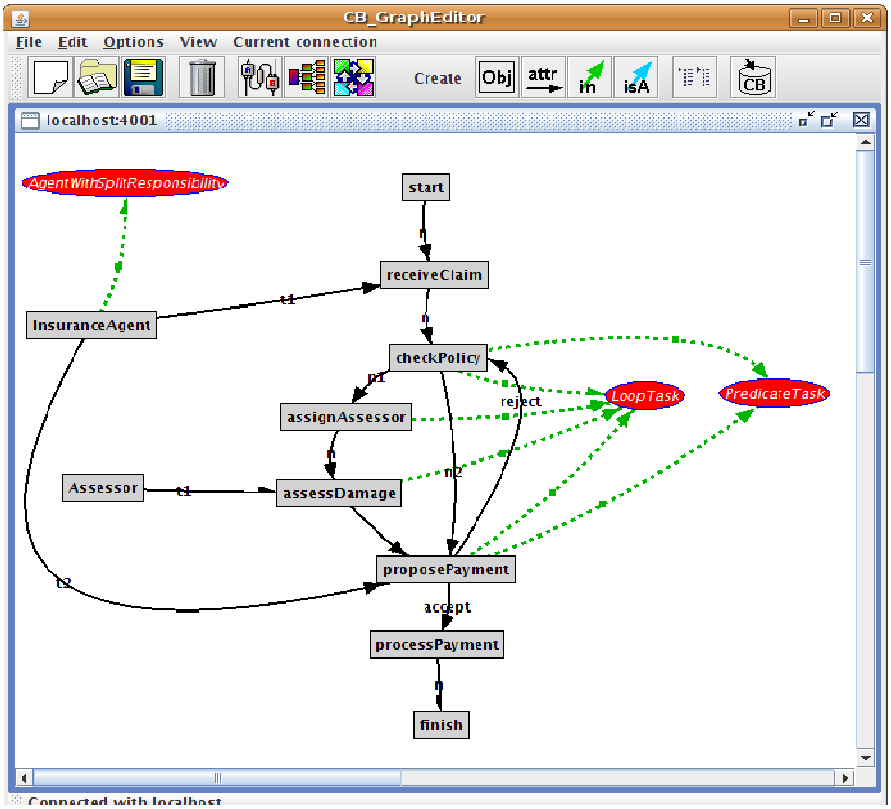

Fig. 2. Graphical analysis of a workflow model

Meta-formulas are made for re-use. For example, we can define the concept of an organizational unit

```
OrgUnit with
   attribute, transitive, asymmetric
     subunit: OrgUnit
end
```

where asymmetry is defined as follows:

```
Proposition in Class with
  attribute
    asymmetric: Proposition
  constraint
    asym_IC: $ forall AC/Proposition!asymmetric
                C/Proposition x,y,R/VAR
              P(AC,C,R,C) and (x in C) and (y in C) and
                     A(x,R,y)  ==> not A(y,R,x) $
end
```

ConceptBase comes with a library of pre-defined meta formulas (multivalued attributes, transitivity, symmetry, etc.) that can be extended and modified, as meta formulas are objects in ConceptBase that can be inserted to and deleted at any time. Other researchers have used this to investigate proposals for new basic abstraction mechanisms in information systems engineering such as materialization [4].

2.3 Active Rules

Active rules (also called event-condition-action or ECA rules) are triggered by an event (e.g. an update), check a condition, and then execute the action part for all variable instantiations of the condition part. They can be used for multiple purposes, e.g. to set initial attribute values whenever an object is created for the first time, or to call an external program upon certain database updates. ConceptBase has a full implementation of active rules. We demonstrate here that it allows to define the execution semantics of Petri nets. Petri nets have places and transitions connected by directed links. Places have a positive number of tokens. A transition is enabled if all input places have at least one token. Firing a transition means to remove tokens from the input places of a transition and to add them to all output places of the transition. The structural part of the Petri net language is expressed in ConceptBase as follows:

```
Place with
  attribute
    sendsToken: Transition
  single
    tokenFill: Integer
end
Transition with
  attribute
     producesToken : Place
end
```

The `tokenFill` attribute is used to define the state of the Petri net. For convenience, we define a function to return the token number of a given place. The function is then used to define the concept of an enabled transition

```
TokenNr in Function isA Integer with
  parameter
   place: Place
  constraint
    c1: $ (place tokenFill this) $
end
EnabledTransition in QueryClass isA Transition with
  constraint
    c1: $ forall pl/Place (pl sendsToken this)
                 ==> (TokenNr(pl) > 0)  $
end
```

A single ECA rule is sufficient to model the execution semantics of Petri nets. We omit here the obvious definitions of auxiliary concepts such as `Connected-Place` and `NetEffectOfTransition`:

```
ECArule UpdateConnectedPlaces with
  mode m: Deferred
  ecarule
        er: $fire/FireTransition tr/Transition pl/Place
                n,n1/Integer
        ON Tell (fire transition tr)
        IF (tr in EnabledTransition) and
         (pl in ConnectedPlace[tr]) and
         (n1 =
  TokenFill(pl)+NetEffectOfTransition(pl,tr))
        DO Retell (pl tokenFill n1)$
End
```

Figure 4 shows a Petri net visualized in the ConceptBase graph editor. The graph editor has been configured to display enabled transitions with a green color. Places with a token are visualized by circles with a corresponding number of black dots.

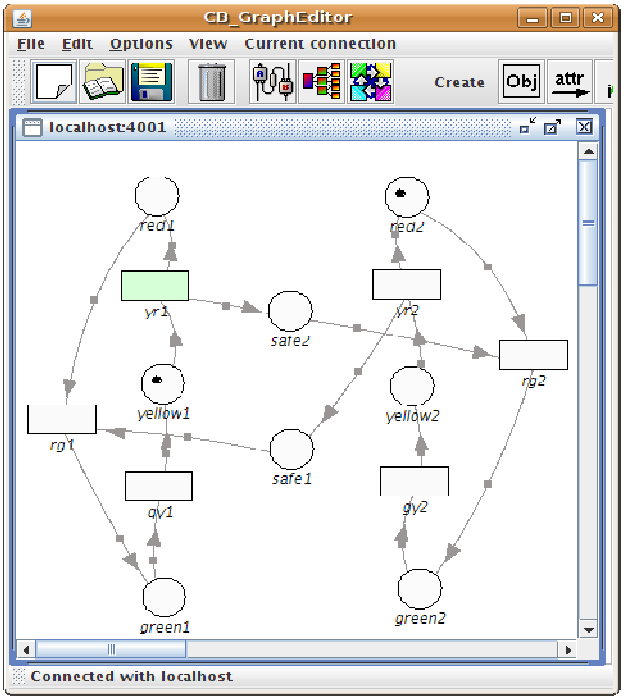

Fig. 3. . Graphical display of a Petri net with ConceptBase

2.4 Function Definitions

The Petri net example illustrates the definition of a simple function `TokenNr`. Functions in ConceptBase are queries that return at most one result per input. As ConceptBase fully supports recursive Datalog, we can reuse this capability to support the recursive definition of certain simple functions. For example, the Fibonacci numbers can be computed by

```
fib in Function isA Integer with
  required,parameter
    n: Integer
  constraint
    cfib: $ (n=0) and (this=0) or
            (n=1) and (this=1) or
            (n>1) and (this=fib(n-1)+fib(n-2)) $
end
```

The definition employs double recursion. A naive evaluation would require exponential time to compute the result. As the second call can reuse the result of the first call, an optimized algorithm requires only linear time. Due to the bottom-up evaluation strategy of Datalog, ConceptBase requires only linear time, i.e. realizes the optimal algorithm with its Datalog engine.

A second example is the computation of the length of the shortest path between two nodes in a graph. This function is useful for a whole family of model metrics. In ConceptBase, this can be defined by a combination of a function and a query definition that call each other recursively:

```
sp in Function isA Integer with
  parameter x: Node; y: Node
  constraint
    csp: $ (x=y) and (this=0) or
           (x nexttrans y) and (x <> y) and
           (this = MIN(spSet[x,y])+1) $
End

spSet in GenericQueryClass isA Integer with
  parameter x: Node; y: Node
  constraint
   csps: $ exists x1/Node (x next x1) and
                          (this=sp(x1,y)) $
End
```

So, the length of the shortest path between `x,y` is 0 iff `x=y`. Otherwise, it is the minimum of the length of all shortest path starting from a successor of `x` plus 1. The function `sp` can be used to define the concept of a node being on a shortest path between two given nodes. In Figure 3, nodes on a shortest path (except the start and end node) are displayed in yellow.

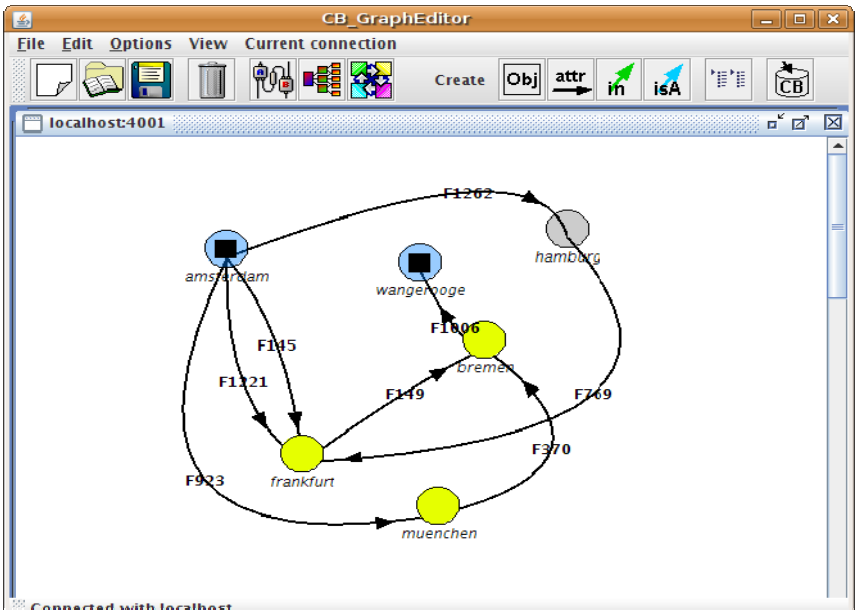

Fig. 4. Graphical representation of nodes on a shortest path

3 Application Experiences and Impact

ConceptBase has been used in a wide variety of application domains where meta modeling and metadata repository management in heterogeneous environments play a role. In most cases, individuals and organizations used ConceptBase to investigate or teach certain concepts, or to prototype ideas from which then code was derived – in a few cases even automatically generated – for commercial systems. Below, we summarize experiences in three broad application areas, namely IS engineering environments, requirements analysis, and the more recent multimedia community management. Detailed descriptions of several applications can be found in [18].

3.1 Repository Management of Heterogeneous Engineering Environments

The original motivation for the development of ConceptBase was the integrated management of requirements [8], Taxis database design specifications [25], and database programs in the European DAIDA project [15]. In a precursor of today's model-driven approaches, semi-automatic tools for the mapping from requirements to designs, and for the code generation from design specs were developed. An important goal was to make this process incremental such that small requirements changes would not lead to a complete repetition of the whole process. This required a meta meta model in which the design objects in the different formalisms, the human design decisions taken in the semi-automatic process, and the tool applications for automated

parts of the process could be documented in a homogeneous manner. This meta meta model was defined and tested in early versions of ConceptBase. In an operational mode, this meta meta model then served as the basis for a query facility by which design tools could store retrieve repository objects under this schema, and by which implications of design changes could be roughly analysed. Constraints were used to prevent tools from inserting inconsistent or incomplete design objects, or to warn against non process-conformant decisions.

Especially the issue of traceability among design decisions spawned a major research initiative in this field which we conducted jointly with researchers in New York and later Monterey and Atlanta, in what was perhaps the worldwide first Internet-based knowledge base management system operating across the Atlantic. In large-scale empirical studies in the US, reference models for different degrees of maturity in traceability were developed using ConceptBase [33], and served as blueprints for the models underlying market-leading traceability tools by Anderson Consulting (now Accenture) and Texas Instrument. Other groups e.g. at the TU Munich used ConceptBase to model the structures of commercial software development environments such as HP's FUSION environment. In our cooperations with engineering groups at RWTH Aachen University, similar repository meta meta models were developed for engineering environments in industrial quality management and in chemical engineering design.

In the European DWQ project on Foundations of Data Warehouse Quality, ConceptBase was employed as an active metadata repository linking models of sources, integrators, data warehouses, and client data perspectives. The repository was used as a semi-shallow documentation mechanism for the inputs and results of description logic reasoners [23, 14], and as a basis for generating code from the metamodel relationships using both local-as-view and global-as-view algorithms [9, 22, 32].

Since 2001. this early work also fed into research on model management conducted at Microsoft Research [1, 2] and influenced our own recent projects on generic metamodels for model management in heterogeneous environments [21, 22]. Such a more active role of the metadata repository was pioneered in a project for a large European software vendor in the mid-1990s where we were able to show that, using a notation-oriented meta meta model and related meta-formulas, the reverse engineering of complex relational databases into entity-relationship models, could be automatically supported to a large percentage with surprisingly little effort [19]. The same turned out to be true for the reverse code analysis of a significant part of one of the world's largest switching systems, Ericsson's AXE system.

3.2 Multi-perspective requirements engineering

Requirements elicitation and management (RE) is well known to be one of the most important and difficult tasks in information systems engineering. An early external example of ConceptBase usage in this field was a requirements analysis tool for Telecommunication Services (RATS) developed as a prototype at British Telecom [5]. Their meta model was quite elaborate, including aspects such as non-functional

quality goals, use cases, and multiple domain models, all coming along with version histories. ConceptBase rules and constraints were used to give some guidance to the development process.

In our own work, we pursued a slightly different line of work. Practice experiences showed that one of the best ways to elicit requirements in complex systems is their capture from many different perspectives – different notations as well as different user task perspectives. Capture is interleaved with inconsistency analyses among these perspectives to spawn debate, thus clarifying mutual misunderstandings and bringing to light hidden assumptions and requirements. This approach which became popular as Viewpoint Analysis in the late 1990's [6, 30], is of course a perfect application example for ConceptBase. Together with the German software and consulting firm USU, we developed a process analysis meta meta model focusing on task interrelationships and media breaks, which was applied successfully in numerous business and software requirements analyses [29]. This application was also the motivation for adding modules to ConceptBase [28].

In the last years, multi-perspective modeling has been extended to the analysis of inter-organizational networks and even of Internet communities, with particular emphasis on rich models of trust evolution in such networks [7].

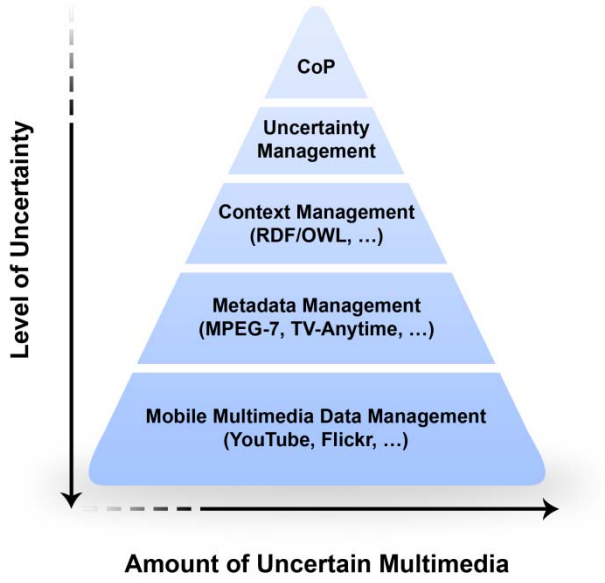

Fig. 5. Modeling perspectives on multimedia community metadata management

3.3 Multimedia Information Engineering

Since the late 1990's, the KBS Hyperbook project at TU Hannover [17, ch. 5] pioneered the idea to use the metadata management facilities of ConceptBase for the structuring of eLearning environments. These experiments also formed a starting point for research in the well-known peer-to-peer learning environment Edutella [27].

In interdisciplinary cooperation with various kinds of media scientists, we have extended such approaches to various approaches to the analysis and support of multimedia communities of practice on the Internet in numerous fields of education and research, ranging from Judaic studies to movie sciences to general contributions to cultural reconstruction in former war areas, to multimedia metadata standards such as MPEG-7/21. Goal is supporting the interaction of communities across different types of media and under different negotiated cooperation regimes; social network analyses are augmented by aspects of media usage and by requirements engineering strategies. Fig. 5 illustrates the number of different perspectives to be considered in such environments [3]. The closeness of many ConceptBase features to recent XML and RDF extensions keeps this work rather directly relevant even for people who are not using the system itself.

4 Summary and Outlook

With its distinguishing feature of powerful multi-level metamodel handling under the well understood and efficiently implemented Datalog semantics, ConceptBase has successfully preserved a niche from which some impact could be achieved in many application domains. We feel that the potential of meta-formula management for multi-language, multi-domain or multi-perspective engineering has still not yet been fully utilized. Ongoing work at Tilburg University shows that traceability can be defined as an extremely versatile ConceptBase attribute category which can then be used to automatically generate inconsistency management analyses across notations, ontologies, or tasks, thus automatically generating a surprising number of traceability tasks in engineering projects.

5 References

1. Bernstein, P.A., Halevy, A.Y., Pottinger, R.A.: A vision for management of complex models. ACM SIGMOD Record, 29(4):55–63, 2000.
2. Bernstein, P.A.,. Melnik, S.: Model management 2.0: Manipulating richer mappings. ACM SIGMOD Intl. Conf. on Management of Data, pp. 1–12, Beijing, China, 2007.
3. Cao, Y., Klamma, R., Jarke, M.: Mobile multimedia metadata management for Virtual Campfire – The German Excellence Cluster UMIC. Submitted for publication, 2009.
4. Dahchour, M.: Formalizing materialization using a metaclass approach. In: CAiSE 98, Pisa, Springer LNCS 1413, 1998.

5. Eberlein, A., Halsall, F.: Telecommunication service development: a design methodology and its intelligent support. Engineering Applications of Artificial Intelligence 10, 6:647-663, 1997.
6. Feather, M.S., Fickas, S.: Coping with requirements freedoms, Intl. Workshop on the Development of Intelligent Information Systems, pp. 42-46, Niagara-on-the-Lake, Ontario, Canada, 1991.
7. Gans, G., Jarke, M., Kethers, S., Lakemeyer, G.: Continuous requirements engineering for organization networks: a (dis-)trust-based approach. Requirements Eng. J. 8, 1: 4-22, 2003.
8. Greenspan, S., Borgida, A, Mylopoulos, J.: A requirements modelling language and its logic. Information Systems 11, 1: 9-23, 1986.
9. Halevy, A.Y.: Answering queries using views: a survey. VLDB Journal 10, 4:270–294, 2001.
10. Hernandez, M.A., Miller, R.J., Haas, L.M.: Clio: A semi-automatic tool for schema mapping. ACM SIGMOD Conf., p. 607, Santa Barbara, CA, 2001.
11. ISO/IEC International Standard, Information Resource Dictionary System (IRDS) – Framework, ISO/IEC 10027, 1990.
12. Jarke, M., Eherer, S., Gallersdörfer, R., Jeusfeld, M.A., Staudt, M.: ConceptBase – a deductive object base for meta data management. J. Intelligent Information Systems 4, 2: 167-192, 1995.
13. Jarke, M., Jeusfeld, M.A., Quix, C., Vassiliadis, P.: Architecture and quality in data warehouses: an extended repository approach. Information Systems 24, 3: 229-253, 1999.
14. Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: Fundamentals of Data Warehouses. Springer, 2nd edn. 2003.
15. Jarke, M., Mylopoulos, J., Schmidt, J.W., Vassiliou, Y.: DAIDA – an environment for evolving information systems. ACM Trans. Information Systems 10, 1: 1-50, 1992.
16. Jarke, M., Rose, T: Managing knowledge about information systems evolution. ACM SIGMOD Conf., Chicago, IL, 303-311, 1988.
17. Jeusfeld, M.A.: Update Control in Deductive Object Bases, PhD Thesis, University of Passau (in German), 1992.
18. Jeusfeld, M.A., Jarke, M., Mylopoulos, J. (eds.): Meta Modeling for Method Engineering. MIT Press, 2009 (in press)
19. Jeusfeld, M.A., Johnen, U.: An executable meta model for re-engineering of database schemas. Intl. J. Cooperative Information Systems 4, 2-3: 237-258, 1995.
20. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+ -- a fully configurable multi-user and multi-tool CASE and CAME environment. Proc. CAISE 96, (Heraklion, Greece), pp. 1-21, Springer LNCS 1080, 1996.
21. Kensche, D., Quix, C., Chatti, M.A., Jarke, M.: GeRoMe – a generic role based meta model for model management. J. Data Semantics VIII: 82-117, 2007.
22. Kensche, D., Quix, C., Li, X. Li, Y., Jarke, M.: Generic schema mappings for composition and query answering. Data & Knowledge Eng. 68, 7:599-621, 2009.
23. Lenzerini, M.: Data integration: a theoretical perspective. 21st ACM Symp. Principles of Database Systems (PODS), pp. 233-246, Madison, Wisconsin, 2002.
24. Melnik, S., Rahm, E., Bernstein, P.A.:. Rondo: a programming platform for generic model management. ACM SIGMOD Intl. Conf. Management of Data, pp. 193–204. San Diego, CA, 2003.

25.Mylopoulos, J., Bernstein, P.A., Wong, H.K.T.: A language facility for designing interactive database-intensive applications. ACM Trans. Database Syst. 5, 2: 185-207, 1980.
26.Mylopoulos, J., Borgida, A.,. Jarke, M., Koubarakis. M.: Telos -- representing knowledge about information systems, ACM Transactions on Information Systems 8, 4: 325-362, 1990.
27.Nejdl, W. et al.: Edutella – a networking infrastructure based on RDF. In: Proc. 11th WWW Conf, Honululu, Hw, pp. 604-615, 2002.
28.Nissen, H.W., Jarke, M.: Repository support for multi-perspective requirements engineering. Information Systems 24, 2: 131-158, 1999.
29.Nissen, H.W., Jeusfeld, M.A., Jarke, M., Zemanek, G.V., Huber, H.: Managing multiple requirements perspectives with metamodels. IEEE Software 13, 2: 37-48, 1996.
30.Nuseibeh, B., Kramer, J., Finkelstein, A.: A framework for expressing the relationships between multiple views in requirements specifications, IEEE Trans. Software Eng. 20, 10: 760-773, 1994.
31.Object Management Group: Meta Object Facility /MOF) core specification version 2.0. OMG 2006.
32.Quix, C.: Metadata Management for Quality-Oriented Information Logistics in Data Warehouse Systems (in German). Ph.D. Thesis, RWTH Aachen University, Germany, 2003.
33.Ramesh, B., Jarke, M.: Reference models for requirements traceability. IEEE Trans. Software Eng. 27, 1: 58-93, 2001.
34.Staudt, M., Jarke, M.: Incremental maintenance of externally materialized views. Proc. VLDB 1996, Mumbai, India, 75, 86, 1996.
35. Staudt, M., Jarke, M.: View management support in advanced knowledge base servers. J. Intelligent Information Systems 15, 3 (2000): 253-285.