REPOSITORY SUPPORT FOR MULTI-PERSPECTIVE REQUIREMENTS ENGINEERING

Hans W. Nissen and Matthias Jarke

RWTH Aachen, Informatik V, Ahornstr. 55, 52072 Aachen, Germany

Abstract — Relationships among different modeling perspectives have been systematically investigated focusing either on given notations (e.g. UML) or on domain reference models (e.g. ARIS/SAP). In contrast, many successful informal methods for business analysis and requirements engineering (e.g. JAD) emphasize team negotiation, goal orientation and flexibility of modeling notations. This paper addresses the question how much formal and computerized support can be provided in such settings without destroying their creative tenor. Our solution is based on a novel modeling language design, M-Telos, that integrates the adaptability and analysis advantages of the logic-based meta modeling language Telos with a module concept covering the structuring mechanisms of scalable software architectures. It comprises four components: (1) A modular conceptual modeling formalism organizes individual perspectives and their interrelationships. (2) Perspective schemata are linked to a conceptual meta meta model of shared domain terms, thus giving the architecture a semantic meaning and enabling adaptability and extensibility of the network of perspectives. (3) Inconsistency management across perspectives is handled in a goal-oriented manner, by formalizing analysis goals as meta rules which are automatically customized to perspective schemata. (4) Continuous incremental maintenance of inconsistency information is provided by exploiting recent view maintenance techniques from deductive databases. The approach has been fully implemented as an extension to the ConceptBase meta database management system and is currently experimentally applied in the context of business analysis and data warehouse design.

Key words: Distributed Requirements Engineering, Meta Modeling, Modular Knowledgr Representation, Inconsistency Management

1. INTRODUCTION

The description of the desired behavior and quality of an information system often becomes too complex for a single person. Recent systems analysis and design methodologies therefore employ multiple partial models or perspectives to structure the set of requirements specifications and designs. However, the methodologies differ substantially in the preferred coupling of partial models and in their consistency management strategies.

Notation-oriented methods manifest their assistance in the set of modeling notations they offer. Examples include structured analysis (e.g. [53]) and object-oriented techniques (e.g. [18]). Typical perspectives include data, behavior, and function, additional structuring principles such as modularization exist often only in very simple form without a well defined semantics.

Domain-oriented analysis methods offer a predefined set of reference models for specific application domains, in addition to the separation of data, function, and behavior, possibly including an organizational and a resource perspective as well. The user can tailor notations, constraints or the contents to the degree foreseen by the developers of the reference models.

Both kinds of methodologies aim at the efficient production of consistent designs across the perspectives they offer. The simplest approach to achieve formal consistency is to avoid redundancy in the modeling approach, i.e. to minimize the overlap of perspectives and thus the risk for what these methods see as modeling 'errors'. Most available CASE tool families are able to deal with predefined sets of such inconsistencies while the ARIS toolset [49] is an example of a tool supporting the domain-oriented approach.

Since the early 1990's, this kind of modeling support has been criticized as neglecting the nature of conflicts as a productive source of creativity. Authors from the software engineering community noticed that multiple stakeholders pursue conflicting opinions, have contradicting requirements and alternative perspectives [4, 17, 14]. In consulting practice, such approaches have been pursued for a long time, albeit without much computer support. Prominent examples include IBM's JAD (Joint Application Design) [3], SSM (Soft Systems Methodology) [11], and PFR (Analysis of Presence and Future Requirements) [1]. Such **goal-oriented teamwork approaches** specifically follow the objective to capture requirements from all available sources and to make arising conflicts productive. They employ moderated teamwork as the main working style which is the reason for the high flexibility and customizability they offer. Their main characteristics are:

Informal information acquisition. The methods employ informal teamwork techniques to acquire information from the participants. Models are developed on drawing or metaplan boards using a set of graphical symbols. The semantics of the symbols are defined by the participants during the workshop, at the time a new symbol is used. Moderators guide this activity and keep the productive atmosphere within the teams alive. The teams typically produce many perspectives in a short time containing lots of conflicts and inconsistensies.

No fixed set of notations. The notations used for information acquisition are not fixed by the methods but instead specified by the participants according to the actual problem to be solved.

Goal-oriented perspective analysis. The analysis of a system and its environment is always focused towards specific goals. These might be the improvement of business processes or the increase of product quality. Goal-oriented teamwork approaches explicitly collect the goals to guide the analysis and resolution of the partial specifications. The specification of a goal may vary from formal to completely informal. Conflicts within a single perspective and between multiple perspectives are analyzed with respect to these goals.

Conflicts, inconsistencies, modeling errors, and gaps between perspectives are detected by a cross-perspective analysis. The results of these comparisons guide subsequent interviews to clarify conflicts and complete the models. The comparisons of the informal, mostly graphical models are performed by moderators and analysts in a manual manner. The situation for the analysts becomes even harder since the notations and the analysis goals can change from one project to another. Experiences show that this manual comparison of perspectives is most often time-consuming, error-prone and incomplete.

Conflict tolerance. Even the existence of conflicts within the final requirements specification are accepted if the reached degree of agreement is adequate to get a consistent system. This is not equivalent to an ignorance of conflicts: all conflicts are recognized and monitored throughout the project.

Changing analysis goals. Viewpoints of stakeholders, parameters of the system environment, and even overall project goals may change quite drastically during the analysis process. The goal-oriented teamwork approaches incorporate the modification of analysis goals as a part of the methodology.

At present, no supporting tools beyond simple groupware tools are available for this class of methodologies. The main reasons are the high degree of customizability the tools must offer, and the lack of formalizations offering the required flexibility.

It is very important to carefully decide which parts of a goal-oriented teamwork approach should be subject to formalization and tool support. The successful, informal team sessions in which the information about the environment and the requirements on the system are acquired, is definitely no adequate part for formal tool support. They seem also not to be the bottleneck. In our opinion, the potential for formalization and computer-based tools lies in another part of the methods: the *cross-perspective analysis* which has so far been performed manually by moderators and analysts to extract knowledge and further questions from the collected information, the partial models.

This paper addresses this problem by providing repository support for goal-oriented inconsistency management in customizable, multi-perspective modeling environments. In the next section we review existing proposals and give an overview of our own approach. In sections 3 and 4 we

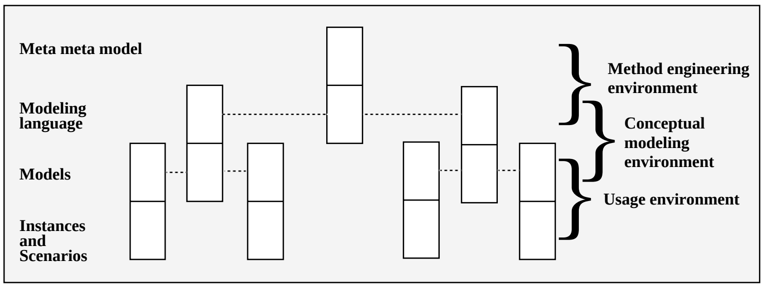

Fig. 1: The ISO IRDS Reference Structure

then present the main contributions. The definition and axiomatization of a modular conceptual modeling language yields a formally precise way of how to define perspectives and control the information exchange between them. On top of this basic formalism, we develop techniques for the definition, compilation, and distribution of analysis goals in a customizable modeling environment (i.e. one without a fixed set of notations or domain models). A distributed execution environment enables efficient incremental maintenance of instance-level information about violations of analysis goals. The paper ends in section 5 with a summary and outlook.

The approach has been implemented as an extension to ConceptBase [26], a deductive meta database manager which uses M-Telos as its object model, and has been evaluated in a number of case studies in business analysis and requirements engineering. For space reasons, some details of proofs and implementation techniques have been left out of this paper; they can be found in [43].

2. EXISTING PROPOSALS AND OUR APPROACH

Summarizing the discussion in the previous chapter, any supporting tool for goal-oriented teamwork approaches should in the first place focus on the cross-perspective analysis. This becomes a problematic task since the tool must not limit the flexibility and customizability of the methods. For this reason, the support of all of the following properties is an essential requirement:

1. Separation of multiple partial models
2. Dynamic customization towards required notations
3. Goal-orientation in perspective analysis
4. Tolerance of conflicts
5. Adaptable definition of analysis goals

This list provides a challenge to all existing proposals: They are only able to handle a subset of these requirements to a sometimes even limited degree. In the next subsection we will take a closer look onto the existing approaches. In subsection 2.2 we present an overview of our approach.

2.1. Related Work

The ISO Information Resource Dictionary System (IRDS) framework [23] provides a general reference structure for multi-perspective (conceptual) modelling. Figure 1 shows its four-layer architecture.

The **Instances and scenarios level** contains objects which cannot have instances. Examples are data, system states, measurements and so on. Objects may have attributes and they may have classes (residing in the model level). During requirements acquisition, when the information system and therefore the instances do not yet exist, this level also contains scenarios of the intended use of the system.

The **Models level** represents the classes of the objects at the instance level. Those classes define the schema (attributes, properties) of the instance level objects as well as rules for manipulating these objects. At the same time the classes are themselves instances of the schema defined at the modeling language level.

At the **Modeling languages level**, meta classes define the structure of the objects (classes) at the model level. In other words, a model is instantiated from the meta classes of the modeling language level.

The **Meta meta model level** contains meta meta classes, with instances at the modeling language level. The definition of multiple modeling languages are possible by appropriate instantiations from these met ameta classes. Moreover, the dependencies between multiple languages can be represented in a static way as attributes between meta meta classes in the meta meta model level, and in a dynamic way as integrity constraints on the meta meta classes.

These four IRDS levels can be grouped into pairs that define interlocking environments, as shown on the right side of Figure 1: usage environments, conceptual modeling environments, and the method engineering environment. In the method engineering environment the common meta meta model is used as the languange to specify the different modeling languages and their interrelationships. In the conceptual modeling environments the modeling languages are used to create conceptual models. The usage environment employs the conceptual models as a schema to manage information of concrete entities.

The interlocking between the models can be read down or up. Reading down, the architecture supports the *generation* of a distributed modeling environment; reading up, it supports the *integration* and *resolution* of existing modelling environments. In either case, the choice of metamodels crucial for the support the model definition and integration environments can offer.

For the integration, it is also important that, at each level, the integration repository can represent information about the separate perspectives from which it is composed (*perspective separation), as well as about their interrelationships and conflicts (*perspective resolution). For both tasks, numerous individual techniques have been proposed.

Separation mechanisms have been developed for different purposes in requirements engineering and for modeling environments. The *Requirements Apprentice* [47] uses a context mechanism called *Cliché* to represent predefined domain descriptions. Since different domain descriptions may be inconsistent to each other, this separation is necessary. They are organised in a specialization hierarchy and can be used as a starting point in requirements engineering. ARIES [27] employs a similar concept called *Folder*. A Folder captures partial domain information and is used for the development of a requirements specification. The engineer creates a new Folder and if applicable a relationship to one of the predefined domain descriptions. He then extends this description according to the actual problem domain. Even though the separation mechanisms have been developed with a somehow different application in mind, they could be employed for the management of multiple perspectives. But all the mentioned approaches do not consider the specification and tolerant evaluation of analysis goals.

Context machanisms have also been studied in Artificial Intelligence (AI). The Cyc knowledge base employs for its huge amount of information a separation mechanism called *context* [22]. Contexts can be organised in a specialization hierarchy which makes the whole contents of the general context accessible in the more specific context. The most specific context, called *BrowsingCntxt*, contains all available informations of the knowledge base and is used to inspect the contents of the knowledge base.

In [40] a general separation mechanism for repositories is presented. Information units of the repository are organised in *contexts*. A context has a unique name and manages its own name space. The same unit may exist in different contexts in different versions. The context can establsh communication channels to share units with other contexts. However, the approach does not incorporate any resolution mechanism and no possibility to specify the goals and the motivation behind the modeling activity. Therefore, inconsistencies between contexts may exist, but will not be detected.

Motschnig-Pitrik [37] develops a set of criteria for evaluating datamodels regarding their ability to represent multiple perspectives. She lists as basic features of a perspective (which she calls a *context*) the unique identifier, operations to populate a perspective, operations to construct a new perspective out of existing ones, the encapsulation of the content, the controlled access to the contents of a perspective, the integration as a first-clas object in the language, and the change propagation. In [38] this approach is applied to the Telos object model. But they do not operate on the logical definition of Telos but only present modifications of the frame syntax intended for user communication. The semantics of these extensions is neither specified nor implemented.

Perspective resolution. The ViewPoint approach [46] is a framework for distributed software engineering, in which multiple perspectives are maintained separately as distributable objects called ViewPoints. The approach aims at a completely distributed architecture without any central control unit. In terms of the IRDS reference structure, no method engineering environment exists. Relationships between ViewPoints are defined by so-called inter-ViewPoint rules. The relationships perform only an integration with respect to the notations; an integration based on the domain under investigation is not supported. The approach does not consider the incorporation of customized notations. The notations are predefined at system definition time.

Many perspective resolution mechanism are limited to a fixed set of analysis goals and in many cases concentrate on syntactical relationships. They operate only on the Conceptual modeling environment and use predefined relations between the modeling languages to analyse perspectives. The integrative view offered by the Method engineering environment is not used. Leite and Freeman employ in [32] such a syntatctic perspective comparison. The contents of perspectives are represented by a set of production rules. They compare the rule bases of two perspectives by identifying the most similar rules as well as the rules with no pairs. On basis of the evaluated mapping they detect wrong, missing and inconsistent information. They do not take information about the domain into account. The analysis rules are predefined in the Static Analyzer and cannot be customized by the user. A tolerant application of the rules is also not supported.

The Carnot project at MCC [12] uses an ontology based perspective resolution mechanism. Carnot employs the general Cyc ontology [33] for the analysis of multiple perspectives. For perspective resolution, the relationships between the perspective and the common ontology are specified. The comparison of these relationships reveals overlappings, gaps and conflicts between perspectives. In terms of the IRDS reference structure, Carnot operates only on the Conceptual modeling environment where all perspectives - also the general ontology - share one modelling language. In [24] the authors describe the role of ontologoes within meta modeling in more detail.

In practice, variants of the IRDS approach have recently migrated into integrative repository products on the one hand, and into generative meta modeling environments on the other. As our solution adopts aspects from both, we shall briefly discuss them below.

Generative meta modeling environments such as MetaEdit+ [29] employ the IRDS structure to support the largely automatic customization towards specific modeling languages. They offer a fixed meta meta model together with a number of predefined consistency-checks which can be employed to specify more complex consistency rules on ontologies and notations. Within the limitation of the set of predefined consistency-checks, project specific analysis goals can be defined in terms of consistency rules on elements of a modeling language. They lack support for the management of multiple perspectives, namely the separation of partial models and the handling of conflicts.

The most comprehensive integrative repository approach in industry is probably the Microsoft repository [6]. On the side of notational and technical integration, it is based on Microsoft's COM data model. As a basis for domain-oriented integration, as far as that exists, a version of Rational's Unified Modeling Language is employed; reference Information Models have been developed by consensus processes with vendors in the form of specializations of this UML information model. There is a fixed meta meta model which is essentially an object-relationship model defined on top of COM. In terms of the requirements defined above, different notations can be supported and, though by programming effort of specialists, refinement is possible. For the separation of perspectives, a workspace concept is offered. Perspective resolution is supported to some degree by the combination of the Relationship concept in the meta meta model with a versioning mechanism. However, support for all these features is at the level of programming rather than declarative statements, even though the task is significantly simplified through types and the clever use of

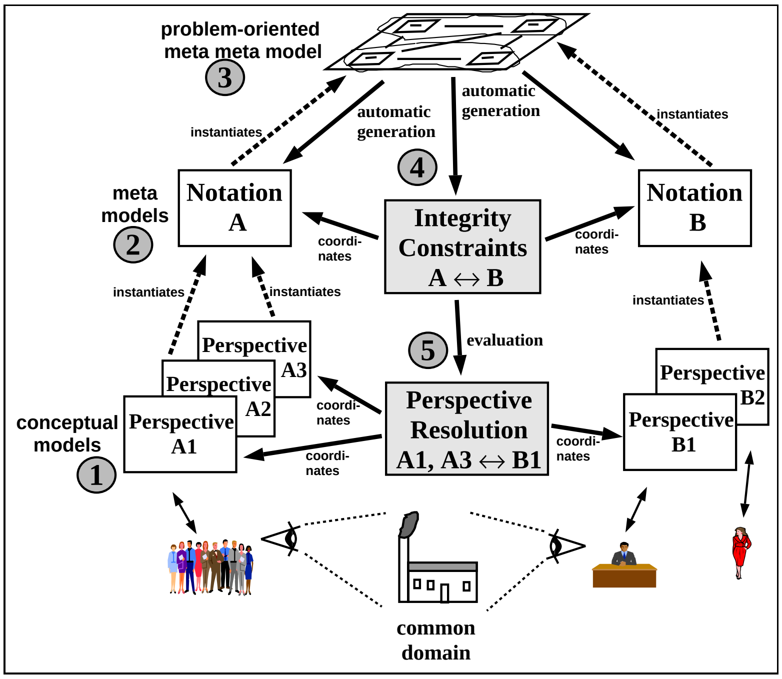

Fig. 2: Overview of the Approach

inheritance.

2.2. Overview of our Approach

Our approach is also based on the IRDS reference structure. Its most prominent characteristic is the management *all levels* within one logical framework. Due to the limitation of SQL, the IRDS reference structure had to treat each level pair as a single entity. Thanks to the extensible meta modeling capabilities of the logic-based conceptual modeling language Telos [39] we are able to manage all four levels uniformly, and to keep even the meta meta model modifiable.

The structure of our approach is presented in fig. 2. We are using only the three upper layers since, in this paper, we are not concerned with the management of concrete entities of the real world, even though these can be easily integrated via scenarios. In contrast to many other approaches (c.f chapter 2.1) we support both, the Conceptual modeling and the Method engineering environment. In the following paragraphs we explain the five features of this approach in more detail.

(1): Separation of Multiple Perspectives.

The conceptual models represent individual perspectives of stakeholders. The figure shows three perspectives (`A1,A2,A3`) expressed in `Notation A` and two perspectives (`B1,B2`) expressed in `Notation B`. Our separation mechanism offers independent modeling contexts and enables the representation of inconsistent conceptual models.

The formal conceptual modeling language Telos [39] forms the basis for our work. In chapter 3 we describe M-Telos which extends Telos by a module concept. Modules act as containers for partial models and enable therefore the separation of conflicting perspectives or viewpoints.

(2): Customizable Notations.

We enable customizable notations by employing the extensible meta modeling capability of Telos. The language allows to define *and* modify meta models of the desired notations. i.e. the notations used to express the partial models are itself specified on the second level, the meta level. The example comprises two notations, `Notation A` and `Notation B`.

(3): Adaptable Specification of Analysis Goals.

A shared meta meta model inter-relates the employed modeling notations. It specifies the domain structure as well as the specific analysis goals. This model is created in teamwork at the beginning of the analysis project and documents an agreed share sub-language of all participating stakeholders. An analysis goal is actually an integrity constraint which defines a cross-perspective check performed on the bottom level between concrete conceptual models.

Since Telos allows for an unlimited classification hierarchy, even the meta meta model is modifiable, i.e. the analysis goals can easily be adapted to specific project objectives.

(4): Goal-Oriented Perspective Analysis.

The analysis goals specified in the problem-oriented meta meta model (cf. no. 3 in fig. 2) are stated independently from any specific notation. They are formulated exclusively on the domain terms. In order to be ables to analyse the partial models residing on the bottom layer, we transform the analysis goals into integrity constraints on the notation meta models. In chapter 4.1 we present an automatic transformation mechanism.

If an analysis goal covers two or more notations, the integrity constraint will be placed in a special module, the resolution module. In such a module (cf. no. 4 in fig. 2) all integrity constraints which specify the relationships between partial models of the connected notations are collected. In the figure we have a resolution module for the two notations mentioned above.

(5): Tolerance of Conflicts

To avoid interrupting the creative modeling activity in the presence of inter-perspective inconsistencies, the cross-perspective analysis takes place in separate resolution modules. The figure shows such a resolution module (cf. no. 5 in fig. 2) to check the perspectives `A1,A3` and `B1`. If the analysis results in a conflict between perspectives, the conflict is continuously documented but it is not required that the conflict is resolved immediately. Since the documentation is visible only within the resolution module, the designers of the involved models are not affected. In chapter 5 we develop a mechanism which is based on deductive database techniques.

3. M-TELOS: SEPARATION OF MULTIPLE PERSPECTIVES

In this section, we describe how a conceptual modeling language and its formal semantics can be extended to address the need for perspective resolution. As stated earlier, our goal is to define an approach that has a sound standard logical basis for conflict analysis, works at all levels of the IRDS hierarchy, and is compatible with standard module notations found in the software architecture literature. As a starting point which already satisfies the first two goals, we choose the conceptual modeling language Telos.

Telos [39] was designed for managing (meta) information about information systems. It integrates aspects from database design, knowledge representation and conceptual modeling.

Like other conceptual modeling languages, Telos offers a textual and a graphical representation. From a user's point of view, the distinguishing feature of Telos in comparison with other conceptual modeling approaches is its extended meta modeling capability. While SQL, for example, is limited to deal with classes and their instances (i.e. exactly one level pair), and most object-oriented languages allow at best one metaclass level, Telos allows the user to manage also the (meta) classes of a class, the (meta meta) classes of a meta class, and so on. The number of these instantiation levels is not limited by the language.

Telos can be used to specify all four levels shown in fig. 3, and even keep the meta meta model modifiable. In the context of requirements engineering from multiple perspective these four levels define different user classes for Telos. Method engineers define the common principles of the domain under investigation on meta meta model level. In the example we are concerned with business processes, where we typically deal with a `medium` that `contains` some kind of `Data`. The Method engineer also specifies the modeling language to be used to formulate the conceptual model. This language comprises then concepts to express the fact that a `Form includes` an `Item`. Model designers use this schema to produce a conceptual model of the problem domain. In the example this model states that there exist a form `Travel Expense Declaration` that includes two items:

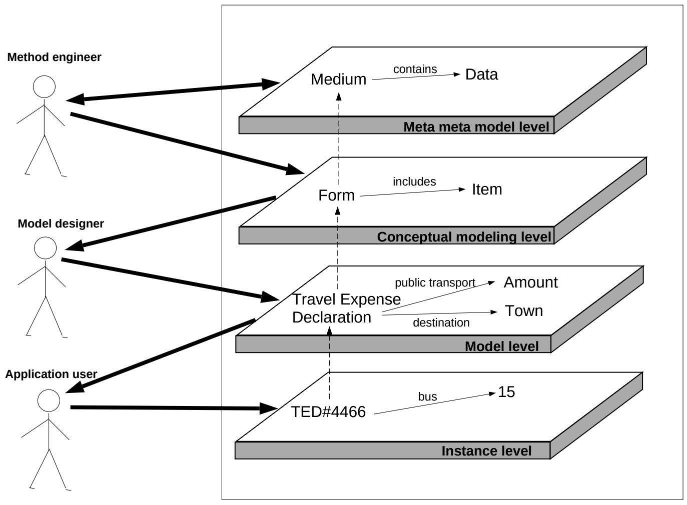

Fig. 3: IRDS and Telos

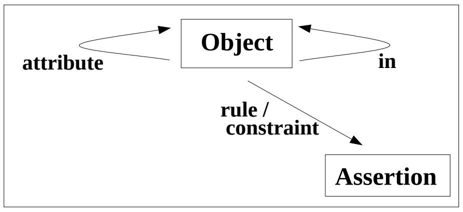

Fig. 4: Telos' builtin objects

the `Amount` of `public transport` expenses and the `desination`. Finally, the Application user inserts concrete entities of the real world, the travel expense declaration `TED#4466` in the example.

A closer look at Figure 3 reveals that any modeling facility supporting such an interlocked way-of-working requires at least three basic language concepts – one for self-standing labeled objects, a second one for labeled links between them, and the third one to express the instantiation relationship between the IRDS levels. In order to provide declarative formal control over the usage of these base constructs, a fourth concept, that of a logical assertion, is also desirable.

As shown in Figure 4, the kernel of the Telos language is just that. All other language facilities (such as generalization hierarchies, cardinality constraints, and so on) can be bootstrapped from this kernel.

In the *textual view* we group together all information for an object (e.g. `TravelExpenseDeclaration`). The class (e.g. `Form`) of that object precedes the object name, the attributes of the object (e.g. `public_transport`) are sorted under *attribute categories* (e.g. `includes`) which refer to the attribute definitions of the object's classes. Note that all objects, i.e. links and nodes, are instances of the builtin object `Object`.

```
Object Form with            Form TravelExpenseDeclaration with
  contains                    includes
     includes: Item              destination  : Town;
end                              public_transport: Amount
                            end
```

Besides inserting and modifying Telos objects (TELL function), the second main function of the server is the ASK facility. Queries are formulated like ordinary classes with a (membership) constraint [52]. They are recognized by the system via the keyword `QueryClass`. The query evaluator computes the answers and establishes an intensional instantiation relationship between the query class and the answers.

The following example presents a query class HighExpenses computing all travel expense declarations with a public transport amount greater than 500. We restrict the set of answers to the declarations by defining the query class as a specialization of TravelExpenseDeclaration. The attributes which should be part of the answer are specified as attributes of the query class. In the example we will get the public_transport attribute for all computed expense declarations. The constraint forms the membership condition, i.e. only expense declarations that satisfy this constraint become answers to the query class. For the example we require that the value of the public transport attribute is greater than 500.

```
QueryClass HighExpenses isa TravelExpenseDeclaration with
  attribute
      public_transport : Amount
  constraint
      c : $ (public_transport > 500) $
end
```

Note that updates (TELL) and queries (ASK) may refer to any abstraction level. Thus, instance level objects are updated and queried in exactly the same way as the concepts of the modeling language level. This is also the basis for perspective resolution at all levels, as shown later in this paper.

Internally, all objects have a unique identifier. However, on the user interface a node object is uniquely identified by its name. To be able to refer uniquely to a link object by only using names, we require that the names of links originating from the same node object are unique. Using the exclamation mark (!) to denote the origin of a link object, the expression TravelExpenseDeclaration!destination refers to the object representing the destination attribute of TravelExpenseDeclaration. For navigation within an assertion two operators are applicable for link objects: from denotes the links origin, to denotes the links target objects. The variable var in literal (TravelExpenseDeclaration!destination to var), for example, would be evaluated to Town.

The ConceptBase user interface includes a customizable graph-browser. The base function is to display node objects like TravelExpenseDeclaration and link objects like TravelExpenseDeclaration!destination. The customization is done by assigning graphical types to nodes and links directly or via deductive rules. It is therefore possible to specify a certain graphical type to all instances of a specific object.

A variant of Telos called O-Telos was formalised in [25] and implemented in ConceptBase [26], a deductive object manager for meta databases. This axiomatization is given in form of a logic program, i.e. each formula is written in form of an integrity constraint or a clause [10]. Therefore, the axiomatization serves two goals at the same time: it provides a formal semantics and, according to the idea of Logic Programming, provides an executable implementation of Telos. To enable large meta databases, a special-purpose object store written in C++ together with numerous optimization techniques speed up this execution.

O-Telos does not contain an explicit mechanism for handling modules even though some features of such a mechanism as well as resolution techniques could be simulated by clever usage of the meta modeling capabilities. However, users found the need for a cleaner and more uniform mechanism akin to the ones found in languages for programming in-the-large. Context mechanisms such as discussed in section 2.1 were found to be either too complicated or designed with other goals in mind. Or the separation into modules would destroy the ability to conduct formally based inconsistency management. Last not least, a design goal for our own language extension, M-Telos, was that for a user who wants to work with just a single perspective nothing should change in comparison with O-Telos. Inconsistency management, as discussed in section 5 of this paper, is just an additional service which the users can employ either in separate resolution modules, or about which they can be made aware using the active database features (notification services) of ConceptBase, in a similar manner as they have already been made aware of integrity constraint violations in previous versions.

M-Telos extends O-Telos by introducing **modules** as a separation mechanism. A module provides an independent modeling context where users can create an individual analysis perspective in the form of a conceptual model. The intended application scenario of modules in concurrent conceptual modeling processes induces the need for **communication** between modules [46]. The module concept supports cooperation among group members by the possibility to define local modules.

Often, one modeling task depends on another one and reuses a part of its results. To support this situation, two modules can communicate by setting up an **import-relationship**. The importing module obtains access to the contents of the imported module. To protect a specific part of the module contents, the concept allows the division of the accessible contents of a module into a private and a public part.

We need not only a modeling context but also a context for the resolution of multiple perspectives. We use dedicated modules for this monitoring task, the so-called resolution modules (cf. section 4). As our experiences indicate [44], such resolution modules need a special way to access the monitored modules. Therefore the module concept offers a **coordination relationship** between modules which enable a resolution module to access all accessible objects of the monitored modules.

For the area of programming languages there exist already a common understanding of modules and their properties [41]. A requirement of the design of M-Telos was to be compatible with these approved principles. As we will show in the next chapters, the M-Telos concept of modules does indeed assure these properties. However, the features of our approach listed in chapter 2.2 go far beyond the capabilities of ordinary modules in programming languages (cf. features 2 to 5).

3.1. Formal Definition of M-Telos

A main goal of the axiomatization of M-Telos was to preserve the **simplicity** of the O-Telos formalization [25]. We had to add only seven new rules and six new constraints to the O-Telos definition. In addition, of course, a number of pre-defined objects had to be defined in order to introduce the built-in module *System* and its relationships to other pre-defined objects. All other M-Telos axioms are just minor adaptations of the O-Telos axioms. The full set of axioms and pre-defined objects is given in the Appendix.

The semantics of a M-Telos object base is given by a mapping to a deductive database containing predefined objects, integrity constraints and deductive rules. This set of objects, constraints and rules constitutes the **axiomatization** of M-Telos.

The basic data structure of M-Telos is very simple. It represents all information using labeled nodes and arcs with object identity.

Definition 1 (Extensional Object Base) Let ID be a set of object identifiers, LAB be a set of labels. An **extensional object base** OB is defined as a finite set of objects:

$$OB \ \subseteq \ \{\, P(o,s,l,d) \mid o,s,d, \in ID,\ l \in LAB\}.$$

Every object is represented in form of a tuple $P(o,s,l,d)$ with object identifier o, start object s, destination object d and label l. The above object o can be read as: "The object s has a relationship called l to the object d". We distinguish four different categories of objects: Objects of the form $P(o,o,l,o)$ are called **individuals**. They represent self-standing entities. Objects containing the special label in like $P(o,s,in,d)$ describe **instantiation relationships**. Objects containing the label isa like $P(o,c,isa,d)$ represent **specialization relationships**. All other objects denote just **attributes**.

Five predefined objects document the four object categories: *Object* contains all objects of an extensional object base as instances; *Individual*, $InstanceOf$, IsA, and *attribute* contain the individuals, instantiation, specialization and attribute relationships as instances (cf. axioms A-1 to A-5).

M-Telos introduces an additional predefined object called *Module* which contains all modules as instances. It offers four attributes: *contains* links to the objects which are defined within the

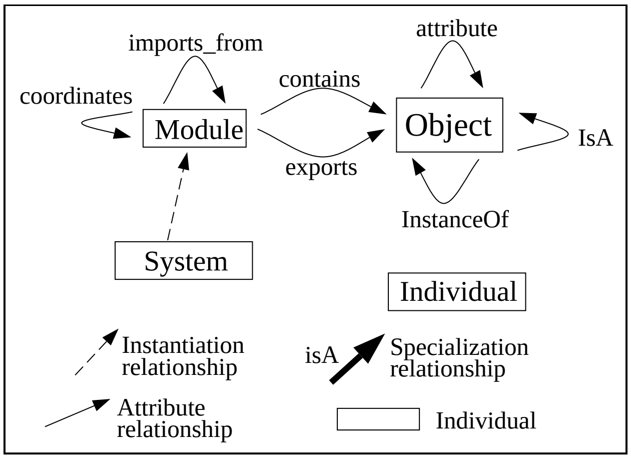

Fig. 5: Predefined objects in M-Telos

specific module; *exports* declares accessible objects to be public; *imports_from* refers to another module and indicates an import relationship; *coordinates* indicates a coordination relationship to another module (cf. axioms A-6 to A-10). A special predefined module called *System* contains all predefined objects including itself (cf. axioms A-11.1 to A-11.26).

Figure 5 visualizes how the new predefined objects extend the semantic network of figure 4. The attributes specifying the contents of *System* are omitted for readability. Individual objects are denoted as nodes of the graph, instantiation, specialization and attribute relationships as dotted, shaded and labelled directed arcs between their source and destination components.

Due to space limitations, we concentrate on the axioms that define the properties of the module concept. The complete list of axioms can be found in appendix A [42].

M-Telos requires that the names of individual objects must be unique within the same module (A-16). Note that names need not be unique among all objects that are *accessible* in a module. Within a single module two individuals with the same name but different object identifiers and different defining modules may exist. Especially for the representation of multiple conceptual models which may be developed by different people observing the same domain this is an essential feature.

The access to objects is only possible through modules, i.e. every object should belong to exactly one module. Axiom A-18 requires a defining module for every object of an extensional object base. In addition axiom A-19 requires that every object is contained in only one module.

We now define the set of accessible objects within a module (as opposed to defined). The new literal $P^{Mod}(M, o, s, l, d)$ describes the objects $P(o, s, l, d)$ which are accessible in module M. The set of accessible objects for a module M comprises

- all objects defined within M (axiom A-22),
- all imported objects (axiom A-23),
- all accessible objects of coordinated modules (axiom A-24), and
- all objects that are accessible in the containing module (axiom A-25).

The closed world assumption (CWA) [35] guarantees that exactly the literals $P^{Mod}(M, o, x, l, y)$ that can be deduced using these rules hold and no others.

The exported objects of a module must form a subset of all the accessible objects of that module. Axiom A-48 formulates this as an integrity constraint. The export part describes a subpart of the conceptual model formed by all accessible objects. This subpart is exported to be reused and extended in another module. To be able to reuse the exported subpart all referenced objects must also be included. This requirement of referential integrity particularly for the export part is formalised in axiom A-49.

For the perspective resolution we shall need the *coordinates* relationship between two modules. The coordination relation is transitive, i.e. if module $M1$ coordinates module $M2$ and $M2$ coordinates module $M3$ then $M1$ also coordintes (indirectly) the module $M3$. This fact is expressed in axiom A-50. Since a cyclic coordination relationship leads to enormous problems in the handling of perspective inter-relationships (see section 4), we explicitly forbid this with axiom A-51.

On basis of the axioms presented so far, other properties of M-Telos are defined which are not directly related to the module concept. We mention in the following only the important ones.

- *Instantiation axiom (A-44):* An instance of a class is allowed to instantiate the class's attributes.
- *Specialization axiom (A-43):* The destination (called superclass) of a specialization relationship inherits all instances of the source (called subclass).

  In combination with the instantiation axiom this defines the attribute inheritance from superclass to subclass: instances of the subclass can instantiate attributes of the superclass.
- *Multiple generalization/instantiation axiom (A-45, A-47):* M-Telos supports multi-classification and multi-generalization under some restrictions.
- *System classes axioms (A-29 to A-39):* For every object the instantiation relationships to the predefined objects $Object, Individual, InstanceOf, IsA$ and *attribute* are deduced and may not be contained in the extensional object base. Also, every instance of the objects $Individual, InstanceOf, IsA$ and *attribute* must have the specific structure introduced at the beginning of this section.

3.2. Properties of the Axiomatization

A set of axioms is of little use if no formal properties can be derived from it that simplify the understanding or the efficient implementation of the language, or form a basis for inconsistency management.

First, we formally define a consistent M-Telos object base as a special deductive database. A deductive database is a triple (EDB, IDB, IC) where EDB, the *extensional database* is a set of facts in the form of relations, IDB, the *intensional database*, is a set of deductive rules defining intensional relations, and IC is a set of closed formulas stating integrity constraints. For a M-Telos object base, EDB becomes the extensional object base containing only facts of the P-relation, IDB is exactly the set of axioms forming deductive rules, and IC is exactly the set of the axioms interpreted as integrity constraints.

Definition 2 (M-Telos Object Base) Let AX_{OB} be all axioms describing predefined objects, AX_R be all axioms which are deductive rules and AX_{IC} be all axioms which are constraints.

Then the triple (OB, AX_R, AX_{IC}) is a **M-Telos object base** if $AX_{OB} \subseteq OB$ holds.

(OB, AX_R, AX_{IC}) is called a **consistent M-Telos object base** if the perfect model of (OB, AX_R) satisfies all integrity constraints of AX_{IC}.

An important property of a consistent object base is the referential integrity within each module. This property guarantees that for every accessible object in a module also the destination and the source components are accessible objects in that module.

Proposition 1 (Referential Integrity [42]) *Let (OB, R, IC) be a consistent M-Telos object base. Then for every object $P(o,s,l,d) \in OB$ and every module M with $P(\#M, \#M, M, \#M) \in OB$ and $In(\#M, \#Module)$ deducible from OB holds: If the object o is accessible in M, i.e. $P^{Mod}(\#M, o, s, l, d)$ is deducible from OB using the rules from R, then also the objects with the identifiers s and d are accessible in M.*

The following proposition formalizes the architecture of a modular knowledge base. The modules always form a tree such that the contents of the System module will be accessible in every single module.

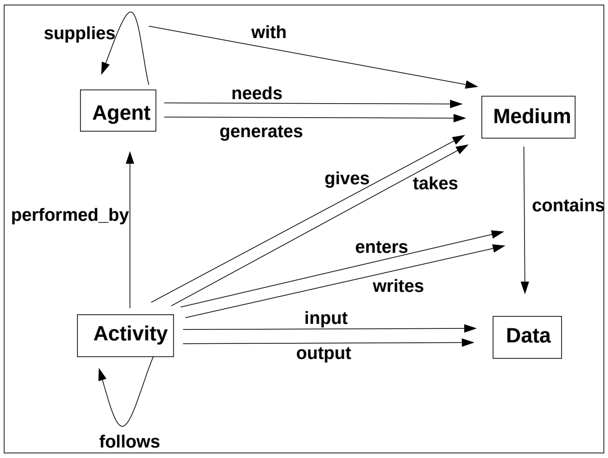

Fig. 6: Example: The PFR meta meta model as specified in M-Telos

Proposition 2 (The *contains* Relation Forms a Unique Tree [42]) *Let (OB, R, IC) be a consistent M-Telos object base. Then (a) all modules are directly or indirectly contained in System and (b) the contains relation forms a tree.*

A distinctive feature of O-Telos is the possibility for unlimited metamodeling. It allows the user to build meta models, meta meta models and so on. This property is preserved in M-Telos. Meta modeling is still possible without any restrictions within a module. In addition the modules can be arranged according to their degree of abstraction. A module then contains only conceptual models of one abstraction level, the more concrete level and the more abstract level reside in different modules. Of course, any combination of these approaches is also possible.

3.3. Case Study: Supporting the PFR Analysis Method

In [44] we reported the application of an early version of our approach to the PFR (Analysis of Presence and Future Requirements) analysis method [1]. We also use this example here to exemplify our approach. We start with a short introduction into the PFR method.

PFR is mainly employed in the early phases of projects developing information systems supporting business processes. The method has three steps:

- In a two-day workshop, stakeholders agree on the scope of the analysis project: the current problems which should be solved and in correspondence to this, the domain structure and the analysis goals. The group also makes a rough analysis of the current business processes in terms of information exchange among organizational units, identifies weak spots and drafts a redesigned business process.
- The perspectives identified as critical to success are then captured in detail by interviews, workflow and document analysis. The acquisition process is accompanied by *cross-perspective analysis* of the captured information for consistency, completeness, and local stakeholder agreement. The results of the comparisons guide subsequent interviews to clarify conflicts and complete the models.
- In a second workshop the goal is to draw together individual perspectives to achieve global *stakeholder agreement.* The step is accompanied and followed by the development of a comprehensive requirements document of typically several hundred pages.

The first workshop leads to a problem-oriented meta meta model defining shared domain terms and analysis goals. Figure 6 presents an M-Telos model that has been used as a default in several PFR analysis projects. It is modified to fit to the actual problems and analysis goals if necessary. The current status of the processes is analysed from three perspectives: the *information-exchange*

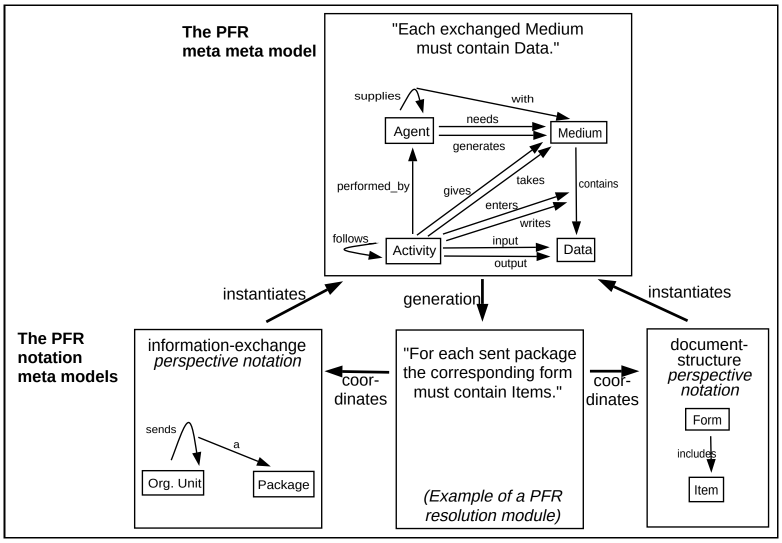
The PFR meta meta model
"Each exchanged Medium must contain Data."
supplies
with
Agent
needs
generates
Medium
performed_by
gives
takes
enters
writes
contains
follows
Activity
input
output
Data
instantiates
generation
instantiates
The PFR notation meta models
information-exchange perspective notation
coor-dinates
"For each sent package the corresponding form must contain Items."
coor-dinates
document-structure perspective notation
sends
a
Org. Unit
Package
(Example of a PFR resolution module)
Form
includes
Item

Fig. 7: Example: The PFR resolution architecture

within the first workshop, the *activity-sequence* and the *document-structure* within the detailed acquisition process in the second step.

The meta meta model in figure 6 explains the basic concepts of these perspectives and their interrelationships. The information-exchange perspective is represented by an `Agent` who `supplies` other agents `with` a `Medium`, the activity-sequence by the `Activity` that is `performed_by` an `Agent` and produces `Data` as `input` or `output`, and the document-structure by a `Medium` that `contains` `Data`.

The meta meta model contains a precise description of the terms that are employed during a PFR analysis. Its structure focuses on the expected problems in the specific domain. The distinction between `Medium` and `Data`, for example, is essential to talk about the unnecessary exchange of documents, i.e. documents which contain data that is not used by any activity.

Figure 7 presents a part of the PFR environment as an example how the two top levels of our architecture (cf. fig. 2) can be represented in M-Telos. The top level module contains the PFR meta meta model together with an analysis goal stating that "Every exchanged Medium must contain Data". This goal formalizes the basic requirement that an efficient business process should not include exchange of documents that do not contain useful data. The formal definition of this goal will be given in section 4.

Below the PFR meta meta model reside the notation meta models to formulate the information-exchange and the document-structure perspectives. The resolution module connected to both notation modules contains the transformed version of the above analysis goal. It specifies an integrity constraint on both perspective notations.

In the next section we will describe how we automatically generate such integrity constraints and how a tolerant consistency management can be implemented on top of M-Telos.

4. CUSTOMIZABLE INCONSISTENCY MANAGEMENT

The main advantage we gain from the Datalog-based formalization of M-Telos is the possibility to customize, in a meta modeling environment, not only perspective ontologies and notations, but also cross-perspective inconsistency management in a declarative manner. This is of particular importance in our application domain, team-oriented requirements engineering.

In contrast to design-level methods such as ER-SA or UML that aim for redundancy-avoidance through concept orthogonality, team-oriented methods such as PFR are designed to provoke the elicitation of conflicts. They employ highly overlapping perspectives and acquire almost all information from different stakeholders (cf. section 1). Due to this redundancy, a large number of relationships exist between perspectives. The cross-perspective analysis checks these relationships. It follows the goals specified in the problem-oriented meta meta model.

In M-Telos, the analysis goals are formulated as so called **meta formulas**: They make statements about objects that reside two abstraction levels below the formula. This is necessary since the perspectives we actually want to analyse reside two levels below the meta meta model (cf. figure 2).

In more technical words: A formula φ is called a meta formula if φ contains a literal (a) $In(x, c)$ where c is a variable, or (b) $A(x, m, y)$ where x is not range restricted by a literal $In(x, c)$ where c is a constant. In such a case we call these literals meta literals.

Example 1 (Meta Formula) The following meta formula is the formalization of the analysis goal given in figure 7 in natural language. The method engineer would formulate such a goal in M-Telos as follows:

```
forall med//Medium, supp//Agent!supplies, with//Agent!supplies!with
(with from supp) and (with to med)
==> exists data//Data, cont//Medium!contains
    (cont from med) and (cont to data)
```

A quantification like `forall med//Medium` defines a variable `med` and requires that its class is an instance of `Medium`, which resides two instantiation levels above `med`.

ConceptBase would then translate this meta formula into an internal logical representation using the axioms defined in the previous section. The resulting internal forms contains several meta literals such as, e.g., $In(med, m')$, $In(supp, s')$, $In(with, w')$. The variables $med, supp, with$ denote objects two levels below the meta meta model. They are not bound to any concrete object on the meta level; m', s', w' are again variables. In this translation process, internal object identifiers are referred to. For readibility, we denote by $\#l$ the object identifier of the object with label l (e.g. $\#Medium$ denotes the object identifier of the object with label $Medium$).

$$\begin{gathered}In(m', \#Medium) \wedge In(d', \#Data) \wedge In(s', \#Agent!supplies) \wedge \\ In(e', \#Medium!contains) \wedge In(w', \#Agent!supplies!with) \wedge In(med, m') \wedge \\ In(supp, s') \wedge In(with, w') \wedge From(with, supp) \wedge To(with, med) \\ \Rightarrow \exists\, data, cont\; In(data, d') \wedge In(cont, e') \wedge \\ From(cont, med) \wedge To(cont, data)\end{gathered}$$

$\square$

Analysis goals for the PFR method exist for a single perspective, for dependencies between multiple perspectives, and to test the desirability of the modeled business processes. Although it is possible to use the analysis goals as they are we will transform them to integrity constraints on the notations's meta models. At this point we have to make clear our terminology in the following subsections: an *analysis goal* is a formula that is specified within the meta meta model and is thus a meta formula. An *integrity constraint* is a formula that is not a meta formula.

The analysis goals represent the agreement among the stakeholders about the goals, or more specific, the questions and problems, the analysis project is dedicated to. Accordingly, our architecture manages the analysis goals within the central module. But the analysis and modeling process does not run in a centralizes way, it is distributed and involves many agents. A central control instance will then be a system bottleneck. On the other hand, a complete distribution without any central control instance like in [46] would not cover the global relationships. To be efficient in such a setting and at the same time be able to manage the global connection we follow the approach of [2] which is a compromise of the two extremes mentioned before: The environments for the perspective development are distributed and work autonomously but there still exists a central instance which has knowledge about their possible inter-relationships. Applied to our case: from the global meta meta model we generate integrity constraints which can be evaluated locally within the modules of the meta level. The global module needs not be accessed during inconsistency monitoring time.

We explain the mechanism in two steps: First we describe the technique of partial evaluation which is used to transform an analysis goal into integrity constraints. Since not all analysis goals need to be transformed to all modules we guide the partial evaluation by generating a transformation plan, i.e. the assignment of analysis goals to modules. The algorithms to compute these plans are subject of the second step.

The last subsection gives then a short overview of the continuous inconsistency management on the instance level. More details on this part are given in [43].

4.1. Partial Evaluation of Analysis Goals

We employ the technique of partial evaluation to transform a meta formula into an integrity constraint. In [25] the application of this technique to the O-Telos object model is presented. We can directly adapt the results to M-Telos. We will therefore only sketch the technique of partial evaluation.

A meta formula contains (meta) literals $In(x, c)$ where c is not a constant but a variable. In our specific case this variable is used to denote an object of the meta model of a notation. The typical situation is thus to have two such literals - $In(x, y)$ and $In(y, c)$ - within one formula where c is an object of the meta meta model and x and y are variables. The meta formula then makes statements about the behaviour of x which denotes an object of a conceptual model. The goal of partial evaluation is to find a solution for y by evaluating the literal $In(y, c)$ within a specific module of the object base. Each occurence of y within the meta formula is then replaced by the computed object. In our case this object comes from the meta level and denotes an object of the meta model of a notation. Since we then already know that the literal $In(y, c)$ evaluates to true with the computed object we can omit the literal and simplify the formula. For every solution of that literal we get a new, partially evaluated version of the original meta formula.

This process has to be repeated for every meta literal until all meta literals have been evaluated. Since we evaluate a meta formula always within a specific module we do it on basis of a notations meta model or a resolution module. The resulting formulas contain no more objects of the meta meta model but only objects of the meta model of a notation. It is therefore only valid for that notation. If not all meta literals of a meta formula can be evaluated in a module then this meta formula is not partially evaluable within this specific module. The implementation of a partial evaluator for Telos is presented in [9].

Example 2 (Transformed Meta Formula) The following formula is the transformed version of the meta formula presented in example 1. In the internal representation, the meta variables are replaced by concrete objects of the two notations for information-exchange and document-structure perspectives (cf. figure 7). The literal $In(med, m')$ of the meta formula is replaced by $In(med, \#Package)$. In addition all the literals connecting these variables to objects of the meta meta model as, e.g., $In(m', \#Medium)$, are eliminated.

$$\begin{gathered}
In(med, \#Package) \wedge In(supp, \#OrgUnit!sends) \wedge In(with, \#OrgUnit!sends!a) \wedge \\
From(with, supp) \wedge To(with, med) \\
\Rightarrow \exists\ data, cont\ In(data, \#Item) \wedge In(cont, \#Form!includes) \\
\wedge From(cont, med) \wedge To(cont, data)
\end{gathered}$$

□

4.2. Computation of Transformation Plans

Unnecessary partial evaluations of analysis goals may arise if modules are connected via *coordinates* relationships: If there exists a *coordinates* link from module A to module B then everything accessible in B is also accessible in A. Any analysis goal that could be transformed for B therefore can also be transformed for A. Since the transformed constraint becomes accessible in A anyway, a separate transformation for A is not necessary. To avoid such inefficiency we compute for every analysis goal the minimal set of modules it must be transformed to.

Algorithm 1 (Computation of Destination Modules) The algorithm first computes the following sets and functions:

- the set M of all modules of the meta level
  $M = \{M_1, \ldots, M_n\}$,
- the set C of all specified *coordinates* relationships:
  $C = \{(M_{1,1}, M_{1,2}), \ldots, (M_{m,1}, M_{m,2})\}$, where $(M_{i,1}, M_{i,2})$ denotes a *coordinates* relationship from $M_{i,1}$ to $M_{i,2}$.
  The set C denotes a directed, acyclic graph.
- the set AG of all analysis goals specified within the meta meta model:
  $AG = \{\varphi_1, \ldots, \varphi_l\}$,
- the function *applicable* which computes for each analysis goal $\varphi \in AG$ the set of modules for which a partial evaluation is possible:
  $applicable : AG \to \wp(M)$
  Whether $M \in applicable(\varphi)$ holds or not is computed by a comparison of the quantifications in φ with the instantiation relationships of M to the meta meta model. Only if for every quantification in φ an instantiation in M exists all meta literals in φ can be evaluated.

```
For every analysis goal $\varphi \in AG$
    compute $P^\varphi = applicable(\varphi)$, the set of potential destination modules
    compute the set $C^\varphi$ with
        $C^\varphi = \{(M_{i,1}, M_{i,2}) : (M_{i,1}, M_{i,2}) \in C, M_{i,1} \in P^\varphi, M_{i,2} \in P^\varphi\}$
    let $E^\varphi = P^\varphi$
    for each $(M_{i,1}, M_{i,2}) \in C^\varphi$
        do
        delete $M_{i,1}$ from $E^\varphi$
        od
```
□

The resulting set E^φ denotes all the modules of the meta level for which a transformation of φ is necessary. The set E^φ is independent from the selections out of C^φ and is always uniquely determined. For E^φ we can prove correctness and completeness concerning the accessibility of transformed integrity constraints as well as its minimality.

Proposition 3 (Correctness and Completeness) *The computed set E^φ for an analysis goal φ is*

(a) correct, i.e. E^φ contains only such modules for which a partial evaluation of φ is allowed.

(b) complete, i.e. the transformation of φ with respect to all modules in E^φ results in the accessibility within all potential destination modules.

Proposition 4 (Minimality) *For every analysis goal φ the algorithm 1 computes in E^φ the minimal set of destination modules such that its accessibility in all potential destination modules is guaranteed.*

Completeness is above defined with respect to the given module structure on the meta level. But there exists a second view on completeness with respect to the analysis goals of the meta meta model: The transformation is complete if all analysis goals of the meta meta model have been transformed.

This kind of completeness does not always hold. If there is no notation for a specific fragment of the meta meta model then the analysis goals specified for this fragment could not be transformed to integrity constraints. The result is that some formal statements represented within the meta meta model could not be tested during the analysis process. In some cases this kind of incompleteness is not a problem or even desired. But in all cases it is useful for the users to get information about analysis goals that can not be transformed.

An automatic completion of the module structure on the meta level is not always possible. A tool could constitute additional resolution modules but cannot automatically establish new notations if a fragment of the meta meta model is not covered yet. We developed an algorithm computing additional resolution modules such that an analysis goal becomes transformable (see [43] for details). The result is in general not minimal. One minimal set can be computed by an algorithm which follows the computation of a minimal set of functional dependencies in relational database schema design [16]: A module is eliminated if its relationships are covered by the reminding modules in the set. We present such an algorithm in [43].

5. CONTINUOUS INCONSISTENCY DOCUMENTATION

In an concurrent modelling activity it is sometimes more profitable not to repair detected inconsistencies immediately but to keep them as *open problems.* Model designers can continue working without spending time in resolving detected errors. The list of unresolved inconsistencies can be handeled in special meetings at certain milestones. In the following we describe our approach in tolerating inconsistencies within a deductive object base.

We call an object base **primary inconsistent** w.r.t. an integrity constraint if the object base violates this constraint. The tolerance of inconsistencies leads to inconsistent objects within a M-Telos object base. But a totally anarchistic state of the object base is not desired. The inconsistent objects must be managed in the sense that the object base knows about the inconsistent objects it contains [17, 48]. We will call objects of the object base that cause an inconsistency **provisionally inserted**. Since we not only have insertions that cause inconsistencies but also deletions, we also have **provisionally deleted** objects.

Once we tolerate provisional objects we get problems with the traditional detection of inconsistencies: It may happen that the current object base satisfies an integrity constraint only because there exist provisional objects [31]. In such a case we call the object base **secondary inconsistent** w.r.t. an integrity constraint. The tolerance of inconsistencies thus comprises two tasks: the detection and the management of provisional objects.

Inconsistencies may occur within a single conceptual model and during the comparison of different models. The ViewPoints approach [46] distinguishes between an in-ViewPoint and an inter-ViewPoint check and employs different techniques to handle them. In our framework we do not need to recognize such a difference and can apply the same mechanisms for both kinds of consistency checks. The check of a single model as well as the check of multiple models takes place within a single module - the cross-perspective check is performed in a resolution module where all models of the coordinated modules are accessible. All the mechanisms we present in this section therefore apply to both the single perspective check and the cross-perspective check.

5.1. *Detection of Inconsistencies*

Like other approaches that deal with inconsistency detection [50, 4, 19] we use rules instead of closed formulas: For an integrity constraint φ we assume an **inconsistency view** $\varphi' \Rightarrow incons_\varphi$ where $incons_\varphi$ is a new predicate symbol and φ' is a closed formula. $incons_\varphi$ is deducible if and only if φ fails, i.e. φ' is the negation of φ.

We call an object that is inserted (deleted) by the current transaction **primary provisionally inserted** (deleted) w.r.t. an inconsistency view $incons_\varphi$ if it takes part in a new computation of $incons_\varphi$. For the formal definition of this property we employ an extension of and-or-trees known from the deductive database area [21, 36]. To do this we have to translate the inconsistency view into an equivalent set of Datalog$^\neg$ clauses [34]. In the most simple case, all and-nodes which are leaves of a new derivation of $incons_\varphi$ and were inserted by the current transaction are then primary provisionally inserted. Due to the use of negation in the body of deductive rules the general case is more complex: We have to change between two object base states in order to evaluate the inconsistency view before and after the transaction. Details of this procedure can be found in [5].

We use a quite similar technique to define secondary provisional objects. Instead of using inconsistency views we now employ **consistency views**: The consistency view of an integrity

detection	*primary*	*secondary*
eager	view maintenance	view maintenance + meta interpreter
on_demand	meta interpreter	meta interpreter

Fig. 8: Detection techniques used in different modes

constraint φ is a rule $\varphi \Rightarrow cons_\varphi$ where $cons_\varphi$ is a new predicate symbol. $cons_\varphi$ is then deducible if and only if the constraint φ is satisfied by the object base.

We again use the and-or-tree for the formal definition of a secondary inconsistency. An object base is secondary inconsistent w.r.t. the integrity constraint φ if the consistency view $cons_\varphi$ is deducible from the object base, but is not deducible from the object base without any provisional objects. In this sense all inserted (deleted) objects of the current transaction that take part in a new derivation of the consistency view $cons_\varphi$ while no derivation without provisional objects exists, are **secondary provisionally inserted** (deleted) objects. Since this definition introduces new provisional objects on the basis of already existing provisional objects we have to compute the least fixpoint of this set.

We distinguish between two detection modes for an integrity constraint: **eager** and **on-demand**. In the eager detection mode the system will check after each transaction for new and for (maybe accidentally) repaired inconsistencies. This mode offers a continuous conflict monitoring (e.g. within a resolution module) which is of course time-consuming and slows down the system. This mode is preferable in a critical development phase (e.g. shortly before an important project review) and when developing critical components (e.g. the control system of an atomic power plant). The eager detection mode is traditionally applied in database area, e.g., [8, 31], and some of the logic-based approaches, e.g., [20]. In [4] the eager mode is also used for software engineering. All the above given definitions are based on an eager detection in the sense that they all were formulated w.r.t. a current transaction.

In the on-demand mode the user controls the timing of the inconsistency check. Here it is not possible to use the current transaction as a filter because no such transaction exists; all objects that take part in the derivation of the inconsistency view will be declared as primary provisionally inserted objects. The detection of provisionally deleted objects is not possible without the concept of a 'current transaction'. The same holds for secondary provisional objects. The on-demand mode is preferable in situations where a fast working style is required, as, e.g., in a brainstorming session. Since the consistency of the database w.r.t. a specific integrity constraint will not be checked after a transaction this mode runs quite fast. On the other hand the users get no idea about the quality of their model. This mode is often used in requirements engineering environments, as, e.g., IPSEN [30] and CONMAN [28]. Also in the ViewPoints approach [46, 15] the user invokes the check of an integrity constraint.

We employ the and-or-tree for the theoretical definition of primary and secondary provisional objects, but not for the implementation of the inconsistency detection. Instead we use efficient incremental view maintenance in combination with a meta interpreter. View maintenance is a machanism that detects all new derivations of a deductive rule or view caused by a transaction. We changed the mechanism developed in [51] such that it returns the whole derivation path.

In eager detection mode, the view maintenance mechanism reports primary inconsistencies. For secondary inconsistencies we must complete the reported derivation trees to include objects that already exist in the object base using the meta interpreter. In on-demand mode, view maintenance is not applicable. We use the meta interpreter to generate the derivation trees where we can detect primary and secondary inconsistencies. Figure 8 summarizes the inconsistency detection mechanisms used in different modes.

5.2. Management of Inconsistencies

Provisionally inserted objects become part of an object base although they violate one or more integrity constraints. For provisionally deleted objects we use a specific feature of Telos: each object has an additional component indicating its *belief time*. The belief time is assigned by the system at transaction time (insertion or deletion). The belief time is a time interval starting at the moment the object was inserted and ending at the moment it is deleted from the object base. Thus, deleted objects are not removed from the object base - instead the object becomes a historical object. Provisionally deleted objects are therefore still accessible. To access historical objects the user must explicitly state that a certain query should be evaluated using the state of the object base at a specific moment.

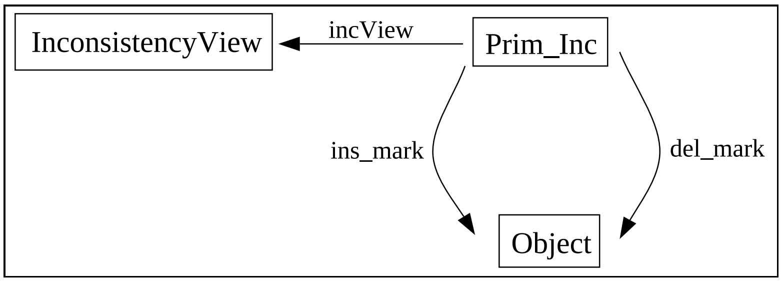

Fig. 9: Marking primary inconsistencies

To avoid a completely anarchical situation, all provisional objects are marked within the object base. The predefined (class) objects indicating a primary provisional object are shown in Fig. 9. The object `InconsistencyView` is a class containing all inconsistency views as instances. Every primary inconsistency, i.e. every validation of an integrity constraint resp. every derivation of the inconsistency view, is identified by a unique instance of `Prim_Inc`. The instance refers to the actual inconsistency view via the attribute `incView` and to the provisional inserted and deleted objects by the attributes `ins_mark` and `del_mark`, resp.

Secondary provisional objects are marked in a similar way. The class structure is given in Fig. 10. In addition to the inserted and deleted objects, we also mark the provisional objects which cause this secondary inconsistency (by attribute `depends_on`). If all these objects would loose their provisional status and become 'normal' objects, the secondary inconsistency would be repaired.

The management of inconsistencies in the eager detection mode does in addition include the detection of accidentally repaired inconsistencies. We therefore store for every provisional object a part of the derivation tree it takes part in as the justification for its provisional status. The justification tree consists of all literals (not their rules) on the path from the marked object to the root and the direct children of positive nodes. If one of these literals is no longer deducible from the object base, the derivation is destroyed and the justification for the provisional status is no longer

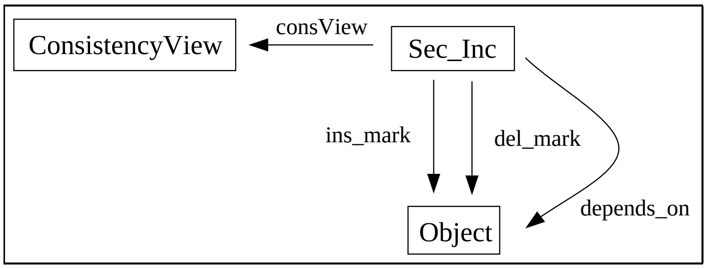

Fig. 10: Marking secondary inconsistencies

valid and the object becomes a 'normal' member of the object base. This technique is similar to reason maintenance systems (RMS) [13] which manage justifications of deduced information.

Everytime an object looses its provisional status we also check the secondary inconsistencies. For every secondary inconsistency we store all the provisional objects that cause this inconsistency. In case the provisional status of an object is annulled we just have to follow these dependencies, delete them and check if this was the last dependency. In this case we also annul all secondary inconsistencies of that integrity constraint because we found a derivation without any provisional objects.

Violation ?		Consequences
Incons.View	Cons.View	
strong	strong	At this combination a transaction must not introduce a primary or a secondary inconsistency. The integrity constraint must be satisfied by the object base.
strong	weak	At this combination a transaction must not introduce a primary inconsistency - but it can introduce a secondary inconsistency. The object base must satisfy or qualified satisfy the integrity constraint.
weak	strong	This combination allows primary inconsistencies to be introduced but no secondary. The object base must satisfy or violate the integrity constraint.
weak	weak	This combination does not make any restrictions concerning the violation of the integrity constraint: primary and secondary inconsistencies are both allowed. The object base may violate the integrity constraint.

Fig. 11: The four consistency levels

5.3. *User Control*

The user can control the tolerance of inconsistencies in different ways. We already mentioned the possibility to specify the detection mode of integrity constraints. A user can also declare if an integrity constraint may be violated or not. We use the term *strong* for a constraint that must not be violated and *weak* for one that may be violated. We even go one step further and allow the user to distinguish between primary and secondary inconsistencies. For each view we can specifiy the detection `Mode` and the possibility to violate it. The declaration of an inconsistency view as strong means that no primary inconsistencies of the corresponding integrity constraint may exist; for a consistency view this leads to the prohibition of secondary inconsistencies.

The introduction of primary and secondary inconsistencies implies a three valued status of an integrity constraint: *satisfaction* if no primary and secondary violations exist, *qualified satisfaction* if it is secondary violated but not primary, and *violation* if it is primary violated. Figure 11 summarizes the four different consistency levels we then can specify for an object base w.r.t. an integrity constraint.

6. CONCLUSIONS AND FURTHER WORK

For some years software specification and design methods have been formalized by a transformation to well-understood formalisms like logic, graph grammars or algebraic specifications to enable a computer-based analysis. It is characteristic of these approaches to assign the methods a fixed semantics the user must accept when using such a system. Beyond that it is assumed that the various partial conceptual models form views on a consistent entire model.

In some other parts of practice just the opposite trend can be observed. Informal teamwork methods leave the details of notations to a great extent to the user and consciously employ conflicts and inconsistencies as an analysis tool, instead of avoiding them. These methods (examples are JAD, SSM and PFR) enjoy increasing popularity exactly because they give negotiation and mutual learning priority over a fixed axiomatization or restriction by reference models. To enhance analysis quality and efficiency formalization and computer support is also desireable for these methods, but they must offer features different to the approaches mentioned above.

In this paper, we developed a comprehensive solution for a computer-based support of team- and goal-oriented analysis methods. We extended the formal conceptual modeling language Telos by a separation mechanism called modules, which enables the representation of multiple, conflicting perspectives. We showed that a simple axiomatization of the extended language M-Telos exists, which allows for a realization by well-understood deductive database technology.

We developed the model-based perspective resolution where the knowledge about the structure of the domain and the analysis goals are specified in a meta meta model. The use of M-Telos as representation formalism keeps even the meta meta model customizable. By declaring the notations as partial views on this model we define a connection between the semantic domain description and the syntactic perspective schemata. We used this connection for goal-oriented inconsistency management by the transformation of domain-oriented analysis goals into notation-based integrity constraints. Since in many cases the simple evaluation of cross-perspective relationships is not enough, we developed a technique for continuous maintenance of inconsistency information based on deductive database technology.

Our approach is completely implemented in ConceptBase [26], a deductive meta database manager which uses M-Telos as object model. In cooperation with the German consulting house and software firm USU we applied our approach in several case studies to business process engineering with the PFR analysis method [45]. The indirect support of the formalization and the computer-based support increased the efficiency of the cross-perspective analysis and the quality of analysis results. Other applications, with different meta meta models, have been developed for industrial quality management, and for the management of development cooperations between small and medium enterprises.

The components of our approach (the axiomatization of a modular knowledge representation language, the goal-oriented perspective analysis, the adaptable specification of analysis goals, and the continuous inconsistency management) can also be used in stand-alone mode together with existing modeling environments or viewpoint mechanisms. The simple axiomatization of the module concept enables its adaption to existing, even non-Telos repositories (as, e.g., the Microsoft Repository [7]) to represent multiple development perspectives. The combination of the goal-oriented inconsistency management concept with notation-centered CASE tools lead to a more guided modeling process with customizable, domain-oriented integrity constraints. This also works for distributed environments like the ViewPoints approach. Since the constraints are checked locally as before, the central goal definition implies no decrease of system performance.

A limitation of M-Telos is that it does not include systematic support for reasoning about concurrent plans which may influence the inconsistency analysis. A logical next step is therefore the investigation of an agent-based extension of M-Telos, nick-named Tropos, which will enable the static and dynamic analysis of the interplay of agent submodels.

Acknowledgment: This work was supported in part by the European Commission under ESPRIT projects CREWS and DWQ, and by the German DFG under the Collaborative Research Center IMPROVE. We would like to thank Manfred A. Jeusfeld and Martin Staudt for their contributions to this work. Thanks are also due to our students Lutz Bauer, Farshad Lashgari, Thomas List, Rene Soiron and Christoph Quix for implementing the system. Numerous suggestions by the anonymous referees and by the editor of this special issue improved the presentation.

REFERENCES

[1] P. Abel. Description of the USU-PFR analysis method. Technical report, USU GmbH, Möglingen (1995).

[2] S. Abitebuol, S. Cluet, and T. Milo. A database interface for file updates. In M.J. Carey and D.A. Schneider, editors, *Proc. of the 1995 ACM SIGMOD Intl. Conf. on Management of Data*, pp. 386–397 (1995).

[3] J.H. August. *Joint Application Design: The Group Session Approach to System Design.* Yourdon Press, Englewood Cliffs (1991).

[4] R. Balzer. Tolerating inconsistency. In *Proc. of the 13th Intl. Conf. on Software Engineering (ICSE-13)*, pp. 158–165, Austin, Texas (1991).

[5] L. Bauer. Inconsistency managemnent in multi-user modeling environments. Master's thesis, RWTH Aachen (in German) (1996).

[6] P.A. Bernstein, T. Bergstraesser, J. Carlson, S. Pal, P. Sanders, and D. Shutt. Microsoft repository version 2 and the open information model. *Information Systems*, **24**(2) (1999).

[7] P.A. Bernstein, K. Harry, P. Sanders, D. Shutt, and J. Zander. The microsoft repository. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases (VLDB)*, pp. 3–12, Athens, Greece (1997).

[8] A. Borgida. Language features for flexible handling of exceptions in information systems. *ACM Trans. on Database Systems*, **10**(4):565–603 (1985).

[9] A. Brogi, S. Contiero, M. Jeusfeld, R. Soiron, J. Lloyd, and M. Milkowska. Applications of Gödel. In K.R. Apt, M. Jarke, W. Nutt, E. Pedreschi, and D. de Schreye, editors, *Gödel and Parallel Prolog, ESPRIT BRA 6810 (Compulog 2), Deliv. 6.2.1* (1994).

[10] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer Verlag (1990).

[11] P.B. Checkland. Soft systems methodology. In J. Rosenhead, editor, *Rational Analysis for a Problematic World*, pp. 71–100. John Wiley & Sons, Chichester (1989).

[12] C. Collet, M.N. Huhns, and W.-M. Shen. Resource integration using a large knowledge base in carnot. *IEEE Computer*, pp. 55–62 (1991).

[13] J. Doyle. A truth maintenance system. *Artificial Intelligence*, **12**:231–272 (1979).

[14] S.M. Easterbrook. Learning from inconsistency. In *Proc. of the 8th Intl. Workshop on Software Specification and Design*, Schloss Velen, Germany (1996).

[15] S.M. Easterbrook and B.A. Nuseibeh. Using viewpoints for inconsistency management. *Software Engineering Journal*, **11**(1):31–43 (1996).

[16] R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Company (1994).

[17] M.S. Feather and S. Fickas. Coping with requirements freedoms. In R. Balzer and J. Mylopoulos, editors, *Proc. of the Intl. Workshop on the Development of Intelligent Information Systems*, pp. 42–46, Niagara-on-the-Lake, Ontario, Canada (1991).

[18] M. Fowler and K. Scott. *UML Destilled: Applying the Standard Object Modeling Language.* Addison-Wesley (1997).

[19] P. Fraternali and S. Paraboschi. A review of repairing techniques for integrity maintenance. In *Proc. of the 1st Intl. Workshop on Rules in Database Systems*, pp. 333–346, Edinburgh, Scottland (1993).

[20] D. Gabbay and A. Hunter. Making inconsistency respectable: A logical framework for inconsistency in reasoning, part i - a position paper. In Ph. Jorrand and J. Kelemann, editors, *Int. Workshop on Fundamentals of Artificial Intelligence Research (FAIR'91)*, pp. 19–32. LNAI 535, Springer-Verlag (1991).

[21] U. Griefahn and S. Lüttringhaus. Top-down integrity constraint checking for deductive databases. In D. H. Warren and P. Szerdei, editors, *Proceedings of the 7th International Conference on Logic Programming (ICLP '90)*, pp. 130–146. MIT Press (1990).

[22] R.V. Guha. Micro-theories and contexts in cyc part i: Basic issues. Technical Report MCC Technical Report No. ACT-CYC-129-90, MCC Microelectronics and Computer Technology Corporation (1990).

[23] ISO/IEC International Standard. *Information Resource Dictionary System (IRDS) - Framework ISO/IEC 10027* (1990).

[24] M. Jarke, K. Pohl, K. Weidenhaupt, K. Jyytinen, P. Marttiin, J.-P. Tolvanen, and M. Papazoglou. Meta modeling: A formal basis for interoperability and adaptability. In B. Kramer, M. Papazoglou, and H.-W. Schmidt, editors, *Information System Interoperability*, chapter 9, pp. 229–263. John Wiley Inc. (1997).

[25] M.A. Jeusfeld. *Update Control in Deductive Object Bases.* PhD thesis, University of Passau (in German) (1992).

[26] M.A. Jeusfeld, M. Jarke, H.W. Nissen, and M. Staudt. Conceptbase. In P. Bernus, K. Mertins, and G. Schmidt, editors, *Handbook on Architectures of Information Systems*, chapter 12, pp. 265–285. Springer-Verlag (1998).

[27] W.L. Johnson and D.R. Harris. Sharing and reuse of requirements knowledge. In *Proc. of the 6th Annual KBSE Conference, Syracuse* (1991).

[28] G.E. Kaiser and R.W. Schwanke. Living with inconsistency in large systems. In *Proc. of the Intl. Workshop on Software Version and Configuration Control*, pp. 98–118, Stuttgart. B.G. Teubner (1988).

[29] S. Kelly, P. Lyytinen, and M. Rossi. Advanced information systems engineering. In *Proc. CAiSE*, number 1080 in LNCS, pp. 1–21. Springer (1996).

[30] M. Lefering. An incremental integration tool between requirements engineering and programming-in-the-large. In *Proc. of the 1st Intl. Symposium on Requirements Engineering*, pp. 82–89 (1993).

[31] P. Leikauf. *Consistency Ensurance by Management of Inconsistencies.* Diss. ETH Nr. 9208. ETH Zürich, (in German) (1990).

[32] J.C.S.P. Leite and P.A. Freeman. Requirements validation through viewpoint resolution. *IEEE Transactions on Software Engineering*, **17**(12):1253–1269 (1991).

[33] D.B. Lenat and R.V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project.* Addison-Wesley, Reading, Mass. (1990).

[34] J.W. Lloyd and R.W. Topor. Making Prolog more expressive. *Journal of Logic Programming*, **1**(3):225–240 (1984).

[35] J. Minker, editor. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers, Inc. (1987).

[36] G. Moerkotte and P.C. Lockemann. Reactive consistency control in deductive databases. *ACM Transactions on Database Systems*, **16**(4):670–702 (1991).

[37] R. Motschnig-Pitrik. An integrating view on the viewing abstraction: Contexts and perspectives in software development, ai, and databases. *Journal of Systems Integration*, **5**:23–60 (1995).

[38] R. Motschnig-Pitrik and J. Mylopoulos. Semantics, features, and applications of the viewpoint abstraction. In *Proc of the 8th Intl. Conf. an Advanced Information Systems Engineering (CAiSE'96)*, pp. 514–539, Heraklion, Greece. Springer-Verlag (1996).

[39] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, **8**(4):325–362 (1990).

[40] J. Mylopoulos and R. Motschnig-Pitrik. Partitioning information bases with contexts. In S. Laufmann, A. Spaccapietra, and T. Yokoi, editors, *Proc. of the Third Intl. Conf. on Cooperative Information Systms (CoopIS-95)*, pp. 44–54 (1995).

[41] M. Nagl, editor. *Building Tighly Integrated Software Development Environments: The IPSEN Approach.* LNCS 1170, Springer-Verlag (1996).

[42] H.W. Nissen. *Separation and Resolution of Multiple Perspectives in Conceptual Modeling.* PhD thesis, RWTH Aachen, Germany, (in German) (1996).

[43] H.W. Nissen and M. Jarke. Goal-oriented inconsistency management in customizable modeling environments. Technical Report 97-12, RWTH Aachen, Aachener Informatik-Berichte (1997).

[44] H.W. Nissen, M.A. Jeusfeld, M. Jarke, G.V. Zemanek, and H. Huber. Managing multiple requirements perspectives with metamodels. *IEEE Software*, pp. 37–47 (1996).

[45] H.W. Nissen and G.V. Zemanek. Knowledge representation concepts supporting business process analysis. In *Proc. of Reasoning About Structured Objects: Knowledge Representation meets Databases (KRDB-95)* (1995).

[46] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, **20**(10):760–773 (1994).

[47] H.B. Reubenstein and R.C. Waters. The requirements apprentice: Automated assistance for requirements acquisition. *IEEE Transactions on Software Engineering*, **17**(3):226–240 (1991).

[48] W. Robinson and S. Fickas. Supporting multi-perspective requirements engineering. In *Proc. of the IEEE Intl. Conf. on Requirements Engineering*, pp. 206–215, Los Alamitos, California. IEEE Computer Society Press (1994).

[49] A.-W. Scheer. Aris. In P. Bernus, K. Mertins, and G. Schmidt, editors, *Handbook on Architectures of Information Systems*, chapter 24, pp. 541–565. Springer-Verlag (1998).

[50] E. Simon. *Conception et Realisation d'un sous Systeme d'Integrite dans un SGBD Relationnel.* PhD thesis, Universite de Paris VI (1986).

[51] M. Staudt and M. Jarke. Incremental maintenance of externally materialized views. In *Proc. of the 22nd Intl. Conf. on Very Large Data Bases (VLDB'96)*, pp. 75–86, Bombay, India (1996).

[52] M. Staudt, H.W. Nissen, and M.A. Jeusfeld. Query by class, rule and concept. *Journal of Applied Intelligence*, **4**(2):133–156 (1994).

[53] E. Yourdon. *Modern Structured Analysis.* Prentice Hall, Englewood Cliffs, New Jersey (1989).

A. COMPLETE LIST OF M-TELOS AXIOMS

The formulation of the axioms follows the idea of a deductive database: Since all axioms are formulated as integrity constraints (e.g. A-12 and A-17), deductive rule (e.g. A-13 and A-14) or as simple facts (e.g. A-1 to A-11.26), they can be interpreted as the core of a deductive database which provides an implementation of M-Telos. In other words, all M-Telos object bases, interpreted as deductive databases with negation and integrity, must consistently include these axioms.

The following objects define the basic structure of an M-Telos object base:

$$P(\#Obj, \#Obj, Object, \#Obj) \quad \text{(A-1)}$$

$$P(\#Indiv, \#Indiv, Individual, \#Indiv) \quad \text{(A-2)}$$

$$P(\#attr, \#Obj, attribute, \#Obj) \quad \text{(A-3)}$$

$$P(\#Inst, \#Obj, InstanceOf, \#Obj) \quad \text{(A-4)}$$

$$P(\#Isa, \#Obj, IsA, \#Obj) \quad \text{(A-5)}$$

The definition of the object *Module* **is given by the following facts:**

$$P(\#Mod, \#Mod, Module, \#Mod) \quad \text{(A-6)}$$

$$P(\#cont, \#Mod, contains, \#Obj) \quad \text{(A-7)}$$

$$P(\#exp, \#Mod, exports, \#Obj) \quad \text{(A-8)}$$

$$P(\#imp, \#Mod, imports\_from, \#Mod) \quad \text{(A-9)}$$

$$P(\#coord, \#Mod, coordinates, \#Mod) \quad \text{(A-10)}$$

The module *System* **contains all predefined objects:**

$$P(\#Sys, \#Sys, System, \#Sys) \quad \text{(A-11.1)}$$

$$P(\#Sysin, \#Sys, in, \#Mod) \quad \text{(A-11.2)}$$

$$P(\#SysC1, \#Sys, c1, \#Obj) \quad \text{(A-11.3)}$$

$$P(\#SysI1, \#SysC1, in, \#cont) \quad \text{(A-11.4)}$$

$$P(\#SysC2, \#Sys, c2, \#Indiv) \quad \text{(A-11.5)}$$

$$P(\#SysI2, \#SysC2, in, \#cont) \quad \text{(A-11.6)}$$

$$P(\#SysC3, \#Sys, c3, \#attr) \quad \text{(A-11.7)}$$

$$P(\#SysI3, \#SysC3, in, \#cont) \quad \text{(A-11.8)}$$

$$P(\#SysC4, \#Sys, c4, \#Inst) \quad \text{(A-11.9)}$$

$$P(\#SysI4, \#SysC4, in, \#cont) \quad \text{(A-11.10)}$$

$$P(\#SysC5, \#Sys, c5, \#Isa) \quad \text{(A-11.11)}$$

$$P(\#SysI5, \#SysC5, in, \#cont) \quad \text{(A-11.12)}$$

$$P(\#SysC6, \#Sys, c6, \#Mod) \quad \text{(A-11.13)}$$

$$P(\#SysI6, \#SysC6, in, \#cont) \quad \text{(A-11.14)}$$

$$P(\#SysC7, \#Sys, c7, \#cont) \quad \text{(A-11.15)}$$

$$P(\#SysI7, \#SysC7, in, \#cont) \quad \text{(A-11.16)}$$

$$P(\#SysC8, \#Sys, c8, \#exp) \quad \text{(A-11.17)}$$

$$P(\#SysI8, \#SysC8, in, \#cont) \quad \text{(A-11.18)}$$

$$P(\#SysC9, \#Sys, c9, \#imp) \quad \text{(A-11.19)}$$

$$P(\#SysI9, \#SysC9, in, \#cont) \quad \text{(A-11.20)}$$

$$P(\#SysC10, \#Sys, c10, \#coord) \quad \text{(A-11.21)}$$

$$P(\#SysI10, \#SysC10, in, \#cont) \quad \text{(A-11.22)}$$

$$P(\#SysC11, \#Sys, c11, \#Sys) \quad \text{(A-11.23)}$$

$$P(\#SysI11, \#SysC11, in, \#cont) \quad \text{(A-11.24)}$$

$$P(\#SysC12, \#Sys, c12, \#Sysin) \quad \text{(A-11.25)}$$

$$P(\#SysI12, \#SysC12, in, \#cont) \quad \text{(A-11.26)}$$

The oid of an object is unique within the object base:

$$P(o, x_1, l_1, y_1) \wedge P(o, x_2, l_2, y_2) \Rightarrow (x_1 = x_2) \wedge (l_1 = l_2) \wedge (y_1 = y_2) \quad \text{(A-12)}$$

These two derived literals ease the definition of axioms:

$$P(o, i, in, c) \Rightarrow In(i, c) \quad \text{(A-13)}$$

$$P(o, x, l, y) \wedge P(p, c, m, d) \wedge In(o, p) \Rightarrow A(x, m, y) \quad \text{(A-14)}$$

The names of modules are unique within the object base:

$$P(o_1, o_1, m, o_1) \wedge P(o_2, o_2, m, o_2) \wedge In(o_1, \#Mod) \wedge In(o_2, \#Mod) \Rightarrow (o_1 = o_2) \quad \text{(A-15)}$$

All names of individuals are unique within the object base:

$$In(M, \#Mod) \wedge P(o_1, o_1, l, o_1) \wedge P(o_2, o_2, l, o_2) \wedge A(M, contains, o_1) \wedge A(M, contains, o_2) \Rightarrow (o_1 = o_2) \quad \text{(A-16)}$$

Names of attributes are unique in conjunction with the individual's name:

$$P(o_1, s, l, d_1) \wedge P(o_2, s, l, d_2) \wedge In(M, \#Mod) \wedge A(M, contains, o_1) \wedge A(M, contains, o_2) \Rightarrow (o_1 = o_2) \vee (l = in) \vee (l = isa) \quad \text{(A-17)}$$

Objects belong to exactly one module:

$$P(o, x, l, y) \Rightarrow \exists M\ In(M, \#Mod) \wedge A(M, contains, o) \quad \text{(A-18)}$$

$$In(M, \#Mod) \wedge In(N, \#Mod) \wedge P(o, x, l, y) \wedge A(M, contains, o) \wedge A(N, contains, o) \Rightarrow (M = N) \quad \text{(A-19)}$$

The *contains* **relationships are deduced for attributes of a module:**

$$In(M, \#Mod) \wedge P(o, M, l, y) \wedge (In(o, \#cont) \vee In(o, \#imp) \vee In(o, \#exp) \vee In(o, \#coord)) \Rightarrow A(M, contains, o) \quad \text{(A-20)}$$

$$In(M, \#Mod) \wedge P(o, M, l, y) \wedge (P(x, o, in, \#cont) \vee P(x, o, in, \#imp) \vee P(x, o, in, \#exp) \vee P(x, o, in, \#coord)) \Rightarrow A(M, contains, x) \quad \text{(A-21)}$$

The following four clauses deduce the predicate P^{Mod}

which denotes the visible of an object within a module :

$$In(M, \#Mod) \wedge P(o, x, l, y) \wedge A(M, contains, o) \Rightarrow P^{Mod}(M, o, x, l, y) \quad \text{(A-22)}$$

$$In(M, \#Mod) \wedge In^{Mod}(M, N, \#Mod) \wedge P^{Mod}(N, o, x, l, y) \wedge A^{Mod}(M, M, imports\_from, N) \wedge A^{Mod}(N, N, exports, o) \Rightarrow P^{Mod}(M, o, x, l, y) \quad \text{(A-23)}$$

$$In(M, \#Mod) \wedge In^{Mod}(M, N, \#Mod) \wedge P^{Mod}(N, o, x, l, y) \wedge A^{Mod}(M, M, coordinates, N) \Rightarrow P^{Mod}(M, o, x, l, y) \quad \text{(A-24)}$$

$$In^{Mod}(N, M, \#Mod) \wedge In(N, \#Mod) \wedge A^{Mod}(N, N, contains, M) \wedge P(N, o, x, l, y) \Rightarrow P^{Mod}(M, o, x, l, y) \quad \text{(A-25)}$$

The following three clauses deduce the predicates $In^{Mod}, A^{Mod}, Isa^{Mod}$,

which denote classification, attributation, and specialization within a module :

$$P^{Mod}(M, o, i, in, c) \Rightarrow In^{Mod}(M, i, c) \quad \text{(A-26)}$$

$$P^{Mod}(M, o, x, l, y) \wedge P^{Mod}(M, p, c, m, d) \wedge In^{Mod}(M, o, p) \Rightarrow A^{Mod}(M, x, m, y) \quad \text{(A-27)}$$

$$P^{Mod}(M, 0, c, isa, d) \Rightarrow Isa^{Mod}(M, c, d) \quad \text{(A-28)}$$

The following axioms require that each object belongs to the class $\#Obj$**:**

$$P^{Mod}(M, o, s, l, d) \Rightarrow In^{Mod}(M, o, \#Obj) \quad \text{(A-29)}$$

$$In^{Mod}(M, o, \#Obj) \Rightarrow \exists\ s, l, d\ P^{Mod}(M, o, s, l, d) \quad \text{(A-30)}$$

The following axioms require that each object belongs to exactly

one of the four classes $\#Indiv$, $\#Inst$, $\#Spec$, $\#Attr$**:**

$$P^{Mod}(M, o, o, l, o) \Rightarrow In^{Mod}(M, o, \#Indiv) \quad \text{(A-31)}$$

$$In^{Mod}(M, o, \#Indiv) \Rightarrow \exists\ l\ P^{Mod}(M, o, o, l, o) \quad \text{(A-32)}$$

$$P^{Mod}(M, o, i, in, c) \Rightarrow In^{Mod}(M, o, \#Inst) \quad \text{(A-33)}$$

$$In^{Mod}(M,o,\#Inst) \Rightarrow \exists\, i,c\ P^{Mod}(M,o,i,in,c) \quad \text{(A-34)}$$

$$P^{Mod}(M,o,c,isa,d) \Rightarrow In^{Mod}(M,o,\#Spec) \quad \text{(A-35)}$$

$$In^{Mod}(M,o,\#Spec) \Rightarrow \exists\, c,d\ P^{Mod}(M,o,c,isa,d) \quad \text{(A-36)}$$

$$P^{Mod}(M,o,s,l,d) \wedge (o \neq s) \wedge (o \neq d) \wedge (l \neq in) \wedge (l \neq isa) \Rightarrow In^{Mod}(M,o,\#Attr) \quad \text{(A-37)}$$

$$In^{Mod}(M,o,\#Attr) \Rightarrow \exists\, s,l,d\ P^{Mod}(M,o,s,l,d) \wedge (o \neq s) \wedge (o \neq d) \wedge (l \neq in) \wedge (l \neq isa) \quad \text{(A-38)}$$

$$In^{Mod}(M,o,\#Obj) \Rightarrow In^{Mod}(M,o,\#Indiv) \vee In^{Mod}(M,o,\#Inst) \vee In^{Mod}(M,o,\#Spec) \vee In^{Mod}(M,o,\#Attr) \quad \text{(A-39)}$$

The isa relation is a partial order:

$$In^{Mod}(M,c,\#Obj) \Rightarrow Isa^{Mod}(M,c,c) \quad \text{(A-40)}$$

$$Isa^{Mod}(M,c,d) \wedge Isa^{Mod}(M,d,e) \Rightarrow Isa^{Mod}(M,c,e) \quad \text{(A-41)}$$

$$Isa^{Mod}(M,c,d) \wedge Isa^{Mod}(M,d,c) \Rightarrow (c = d) \quad \text{(A-42)}$$

Inheritance of class membership along the specialization relationship is deduced:

$$In^{Mod}(M,i,c) \wedge P^{Mod}(M,o,c,isa,d) \Rightarrow In^{Mod}(M,i,d) \quad \text{(A-43)}$$

Attributes are typed by their attribute classes:

$$P^{Mod}(M,o,s,l,d) \wedge In^{Mod}(M,o,p) \Rightarrow \exists\, c,m,k\ P^{Mod}(M,p,c,m,k) \wedge In^{Mod}(M,s,c) \wedge In^{Mod}(M,d,k) \quad \text{(A-44)}$$

Consistency of the specialization relationship:

$$Isa^{Mod}(M,o_1,o_2) \wedge P^{Mod}(M,o_1,c,l_1,e) \wedge P^{Mod}(M,o_2,d,l_2,f) \Rightarrow Isa^{Mod}(M,c,d) \wedge Isa^{Mod}(M,e,f) \quad \text{(A-45)}$$

The attribute refinement requires refinement of target objects:

$$Isa^{Mod}(M,c,d) \wedge P^{Mod}(M,o_1,c,l,e) \wedge P^{Mod}(M,o_2,d,l,f) \Rightarrow Isa^{Mod}(M,e,f) \wedge Isa^{Mod}(M,o_1,o_2) \quad \text{(A-46)}$$

The multiple classification with identical attribute names requires a common general class:

$$In^{Mod}(M,i,c) \wedge In^{Mod}(M,i,d) \wedge P^{Mod}(M,o_1,c,l,f) \wedge P^{Mod}(M,o_2,d,l,g) \Rightarrow \exists\, e,h,o_3\ In^{Mod}(M,i,e) \wedge P^{Mod}(M,o_3,e,l,h) \wedge Isa^{Mod}(M,e,c) \wedge Isa^{Mod}(M,e,d) \quad \text{(A-47)}$$

The export part of a module must satisfy the referential integrity:

$$In(M,\#Mod) \wedge P(o,x,l,y) \wedge A^{Mod}(M,M,exports,o) \Rightarrow P^{Mod}(M,o,x,l,y) \quad \text{(A-48)}$$

$$In(M,\#Mod) \wedge A^{Mod}(M,M,exports,o) \wedge P^{Mod}(M,o,x,l,y) \Rightarrow A^{Mod}(M,M,exports,x) \wedge A^{Mod}(M,M,exports,y) \quad \text{(A-49)}$$

The *coordinates* **relationship is transitiv, but not cyclic:**

$$In(M,\#Mod) \wedge In(N,\#Mod) \wedge In(P,\#Mod) \wedge A(M,coordinates,N) \wedge A(N,coordinates,P) \Rightarrow A(M,coordinates,P) \quad \text{(A-50)}$$

$$In(M,\#Mod) \wedge In(N,\#Mod) \wedge A(M,coordinates,N) \wedge A(N,coordinates,M) \Rightarrow (M = N) \quad \text{(A-51)}$$