

THE USE OF CONCEPTUAL MODELS DURING THE DESIGN OF NEW TELECOMMUNICATION SERVICES

Ali Roshannejad, Armin Eberlein

Electrical and Computer Engineering Department

University of Calgary

Tel: (403) 220-5002

Fax: (403) 282-6855

Email: {roshanne|eberlein}@enel.ucalgary.ca

ABSTRACT

The increasing demand for new telecommunication services encourages service providers to extend their service offerings. However, development of such services and their deployment on the public network are non-trivial tasks. Experience has shown that the use of conceptual models can assist in the development of new systems. This paper investigates the usefulness of conceptual models containing information on the telecommunications domain and development process models, during the development of new telecommunication services. After a brief introduction to conceptual models, telecommunication services and requirements engineering, informal, semi-formal and formal conceptual models are discussed. Then Telos, an object-oriented knowledge-representation language that is well suited for conceptual modeling, will be introduced. Some examples of the application and integration of informal, semi-formal and formal conceptual models in the Requirements Assistant for Telecommunication Services (RATS) will be given.

1. INTRODUCTION

The demand for new telecommunication services is continuously increasing. Considering the complexity and size of new telecommunication services, a great deal of time and effort is required for the development and management of new services. Thus a systematic development approach is highly desired. However, it is not only telecommunications that struggles with the development of complex systems; so does software engineering in general [1]. In fact, since telecommunication systems and services consist of a large amount of software, many of the problems in telecommunications appear to be inherited from software engineering.

Studying the history of software engineering shows that understanding the requirements and their correct transformation into code pose considerable difficulties [2]. One of the most common sources of dissatisfaction with a newly developed system is the lack of success in meeting the users' needs. In other words, one has to pay particular attention when defining a new service to ensure an accurate understanding of the users' requirements and needs. Requirements are "a specification of what should be implemented". "They are descriptions of how the system should behave, or of a system property or attribute" [3]. Based on these definitions, one needs to have a clear understanding of the users' requirements in order to develop a successful system.

Obviously, it is impossible for a service designer to have an adequate understanding of all the factors to be considered during service development. This is a great impediment to a complete and consistent requirements specification. Conceptual models of the telecommunication domain can alleviate this problem.

After the acquisition of requirements, the issue of representing and reasoning about the collected knowledge is predominant. This is often done using a modeling approach [4].

2. CONCEPTUAL MODELS

Conceptual models are used to help reduce the amount of complexity that must be comprehended at one time. Compared with the cost of major rework, conceptual models are inexpensive. They should not only be created of the system under development, but also of its environment and the development processes. Three different kinds of languages can be used for conceptual modeling: informal, semi-formal and formal modeling languages.

2.1. Informal Modeling Languages

Text and recordings in natural languages, as well as pictures, animations and videos fall into this category. They are very expressive and can easily be understood by all stakeholders, including non-technical customers. However, they are very prone to inconsistencies, contradictions, incompleteness and misunderstandings.

2.2. Semi-formal Modeling Languages

Entity Relationship Diagrams (ERD), Extended Entity Relationship model (EER), Data Flow Diagrams (DFD), and State Transition Diagrams (STD) are commonly used within industry for the semi-formal expression of requirements. Although they are not quite as easy to comprehend as informal languages, they still have intuitive characteristics that can be understood by technical people, and provide a good overview of the system. Such languages represent a good compromise between complete informality and complete formality. They can for instance be used for the transition from informal requirements to a formal specification.

2.3. Formal Models

Formal conceptual modeling languages have been around for quite a while but their use is not that widely spread. They have all the advantages of formal languages (such as conciseness, completeness, automated reasoning, etc.) and some of them are even executable. However, formal languages are very hard to understand and some of them require very detailed mathematical knowledge.

All these three kinds of languages have their advantages and disadvantages. Although the choice of a language is very difficult and dependent on the specific situation, expressiveness, precision and familiarity are the key issues to be considered when making a choice. It seems that a good approach is to integrate informal, semi-formal and formal languages to use the best of each. However, the relationship between semi-formal and formal representations is less well understood and the transformation between them very much depends on the notations used and will require expert knowledge of the notations.

Telos, as an example of a formal conceptual modeling language, is able to represent an informal object (such as a textual description of a system), a semi-formal object (such as a data flow diagram) as well as a formal object (such as a piece of SDL specification).

In order to effectively use a formal conceptual modeling language, the language needs to be integrated into a methodology and implemented in a tool. ConceptBase[®] is a tool that implements Telos in a deductive object base management system specifically intended for meta data management and the coordination

of design environments. It integrates techniques from deductive, object-oriented and temporal databases in the logical framework of the data model, Telos. Thus ConceptBase can be used for the development of conceptual models in the Telos language. For more information about ConceptBase, see [5].

3. TELOS

Telos' structurally object-oriented framework generalizes earlier data models and knowledge representation formalisms, such as Entity-Relationship Diagrams or semantic networks, and integrates them with predicative assertions and temporal information [6]. As discussed briefly in the previous section, Telos is a formal, object-oriented knowledge-representation language, intended to support the development of information systems. It is able to combine informal, semi-formal and formal conceptual modeling languages while itself being a formal language. Telos is an excellent means of integrating the whole range of formality, and of being able to use the advantages of various kinds of notations.

Telos is not a programming language, but a knowledge-representation language that includes a formal notation, a deductive mechanism, clear semantics for the notation and facilities to structure and organize the knowledge. This section intends to give a brief introduction to Telos, which is required for implementing RATS, a knowledge-based system that aids requirements engineers in the development of telecommunication services [7]. A comprehensive description of Telos can be found in [6].

As a hybrid language, Telos supports three different representation formats: a *logical*, a *frame* and a *graphical* representation. The latter two forms of representation are based on the logical one.

3.1. Logical Representation

Knowledge expressed in Telos is stored in a knowledge base (KB). The Telos knowledge base is a finite set of interrelated propositions or objects. Everything is modeled as an object (i.e., classes, relationships and facts). Telos provides predefined classes, such as Class, String, Integer, Real, etc., that the designer can use to build user defined classes, which are also stored as objects in the knowledge base. The format of an object is as follows:

$$KB = P(\text{oid}, x, l, y, tt)$$

where oid (object ID) is the key property within the knowledge base. The above can be read as follows:

The object x has a relationship called l to the object y . This relationship is believed by the system for the time tt .

One of the advantages of Telos is the way it handles time. In fact, “*belief time*”, which is automatically inserted into the proposition by ConceptBase, is the time handling measure. The “*belief time*” is right-open until the object is removed from the knowledge base.

Telos generally identifies four different kinds of propositions and gives them the following names:

- **Individual** defines an object with a name and a believed time.
- **InstanceOf** defines an instance of a class and a believed time. It represents an instantiation of a class.
- **IsA** defines a specialization of a class (subclass creation).
- **Attribute**: All other propositions represent attributes of a class.

3.2. Frame Representation of Telos

The designer does not work with propositions, but uses the Telos frame representation. All the information inserted into the knowledge base is expressed in such frames. It is the task of ConceptBase to generate the propositions from the frames, giving each object a unique identifier. In order to illustrate the usage of the frame syntax, a sample object, **Employee**, can be defined as follows:

```

Individual Employee in Class with
attribute
  name : String;
  salary : Integer;
  dept : Department;
  boss : Manager
attribute, comment
  comment1 : "The Employee is a person who works for a
  certain period of time in a department under supervision of a
  Manager."
end

Individual Manager in Class isA Employee
end

Individual Department in Class with
Attribute
  Head : Manager
end
  
```

Figure 1: Telos frame representation (class definition)

where **Employee**, **Manager** and **Department** are user-defined classes and **String** and **Integer** are predefined classes. In addition to defining the classes, instantiation of each class can be done in a similar manner. Figure 2 presents the syntax for instantiations:

```

Individual John in Employee with
attribute,name
  theName : "John Williams"
attribute,salary
  theSalary : 65000
attribute,Department
  theDept : Electrical
attribute,boss
  theBoss : David
end

Individual David in Manager
end

Individual Electrical in Department with
attribute,Head
  theHead : David
end
  
```

Figure 2: Telos frame representation (class instantiation)

3.3. Graphical Representation

The contents of the Telos knowledge base can be graphically displayed in a way that is similar to entity-relationship diagrams [8]. Objects are shown as nodes, while instantiation, specialization and attribute relationships are represented as dotted, shaded or labeled directed arcs between their sources and their destination nodes. Figure 3 illustrates a graphical representation. The graphical representation provides an excellent tool for viewing the objects, their attributes, their super and subclasses, as well as any existing relationships between them.

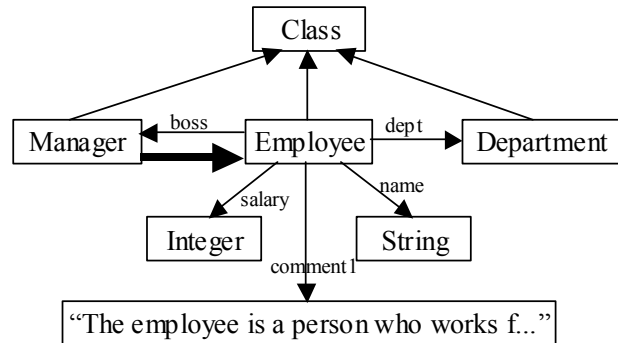


Figure 3: Graphical representation of Telos frames (classification)

3.4. The Predictive Sub-Language of Telos

Telos has a typed first-order assertion sub-language for the specification of integrity constraints, deductive rules and queries. In Telos, both constraints and rules are defined as attributes of classes. This sub-language defines a set of literals that provide access to the Telos knowledge base. Some examples of literals are:

- (x in c) : Object x is an instance of class c.
- (c isA d) : Object c is a specialization (subclass) of d.
- (x m y) : Object x has an attribute to object y and this attribute is an instance of an attribute category with label m.
- From(p,x) : Object p has source x.
- To(p,y) : Object p has destination y.

In addition, this sub-language offers a set of literals that can only be used in formulae. These literals are: <, >, <=, >=, =, <> and ==.

3.4.1. Integrity Constraints

Consistency in conceptual models can be enforced using constraints. Integrity constraints ensure consistency in the knowledge base. Suppose that a manager should not earn less than 70,000. This constraint can be expressed in Telos as follows (see figure 4):

```
Individual SalaryConstraint with
attribute,constraint
constraint1 : $forall m/Manager x/Integer
(m salary x) → (x >= 70000)$
end
```

Figure 4: Integrity Constraint

3.4.2. Deductive Rules

Furthermore, redundancy in models can be reduced using deductive rules. Rules are a perfect tool for adding attributes to a class in a soft-coded manner. For example “if a manager is boss of an employee and the employee works in a department, the manager should be head of the department” (an employee can not be supervised by a manager of another department). This rule is given in figure 5.

```
Individual BossConstraint with
attribute,rule
rule1 : $forall e/Employee d/Department m/Manager
(e dept d) and (e boss m) → (d Head m)$
end
```

Figure 5: Deductive Rule

3.4.3. Queries

In addition to integrity constraints and deductive rules, a conceptual modeling language should provide means for generating queries. If one wants to find all employees who do not have a boss, the following query retrieves them.

```
QueryClass NoBoss isA Employee with
constraint,attribute
con1 : $not (exists m/Manager (this boss m))$
end
```

Figure 6: Queries

3.5. The Model Creation Steps

In order to create a Telos model, the following steps have to be followed:

- *Creation of Telos classes;*
- *Creation of Telos tokens* (specific instances of the classes created in the previous step);
- *Creation of Telos rules* to derive redundant information and to ensure consistency;
- *Creation of Telos constraints* to enforce model consistency;
- *Creation of Telos queries* to allow the retrieval of new information from the knowledge base.

In practice, several iterations of the above steps are required in order to develop a good conceptual model.

4. SOME RATS MODELS

In the case of requirements engineering, conceptual models can be used to help with various issues, such as:

- Facilities for pre- and post-traceability;
- Management of overall development process;
- Intelligent support for the development process;
- Impact analysis of changing requirements;
- Defining and implementing relationships between requirements.

Many of the issues of requirements management are concerned with the relationships between requirements. Therefore, a fundamental task in requirements management is to cross-connect and inter-link requirements with each other. There are three different links between requirements possible.

- Only *one* kind of link with *many* different attributes;
- *a few* kinds of links with *a few* attributes for each link;
- *many* different kinds of links with very *few* attributes per link-type.

In RATS, we used the second approach, which is a compromise between the other two. For the requirements management purposes, two groups of links are defined in RATS, which are: *links with attributes* and *links without attributes*.

Requirements links with attributes have at least two attributes. One attribute states the *reason* for this link and the second attribute gives the *agent* who created the link. Bear in mind that justifying the existence of a requirement or recalling its developer might be needed in the future, especially during maintenance. There are three distinctive requirements links that belong to this first category: *history link*, *logic link* and *structure link*.

History links show the historical development of a requirement, and therefore, provide the means for pre-traceability. Usually, each requirement has at least one history link to the requirement that initiated it. These links are created automatically by the RATS tool and the requirement engineer is prompted to provide the

information required by the attribute template. The template ensures that all the relevant issues are addressed (such as *when*, *where*, *why* and *by whom* the requirement was inserted).

Logic links are similar to history links, however they are more forward-oriented and related to post-traceability. For instance, they are used to link up a goal with the functional requirements and/or the implementation constraints that satisfy it. If it is required to show all the requirements that are logically affected by a change of the requirement concerned, the importance of logic links becomes evident.

Structure links have the task of including and organizing requirements into requirements structures. A requirements structure can be either a document, or any kind of overall use case, a service or a service feature. The structure link works together with the *contains* attribute, which belongs to requirements structures. As soon as a certain requirement is included in a requirements structure, Telos rules ensure that all subsequent requirements with which it has a structure link are automatically included in the structure. Hierarchical ordering is also provided by structure links.

In order to help on a lower layer of requirements management, links without attributes were implemented. Three of these links are *alternative links*, *origin links* and *impact links*.

Alternative links give the possibility of pursuing specification alternatives until a final solution is selected.

Origin links are inserted automatically between each requirement and all those requirements that are predecessors in the chains of history links leading to the present requirement. This link provides comprehensive pre-traceability allowing the user to discover all the requirements that have led to the current requirement from a historical point of view. Knowing where a requirement comes from and why it exists is of critical importance during requirements management.

Impact links are automatically inserted between a requirement and all subsequent requirements in the chains of logic links going out from the present requirement. This link provides comprehensive post-traceability essential for impact analysis and change management.

Development Layer

The RATS development methodology is implemented in the Telos language and is architecturally part of the development layer of the RATS tool. This layer is responsible for the development of a new service by giving advice to the service designer, and by preventing erroneous specifications using various consistency checks. The following is a brief description of one of the three modules in the development layer of RATS.

The Intelligence Module

The main task of this module is to assist the service designer by providing comprehensive guidance during service specification. There are two kinds of guidance, *passive guidance*, which points out any mistakes in the specification, and *active guidance*, which tells the service designer what to do next.

Passive guidance does not tell the service designer what has to be done, but points out inconsistencies. Rigid constraints are used to prevent insertion of inconsistent objects into the knowledge base. Telos provides rigid constraints in three forms:

- Object-orientation (using concepts like inheritance, classification, specialization and instantiation);
- Telos axioms (e.g., syntactical rules of the language);
- Permanent constraints and rules (prevention of insertion of an object to the knowledge base that would violate the constraints).

In addition, temporary user-defined constraints and rules, as well as meta-attributes and state-classes can be included in Telos models.

What passive guidance can do is limited to pointing out errors, but it cannot show what needs to be done. In other words, passive guidance is a negative response when the user violates certain rules and constraints. The following gives an example of passive guidance in RATS:

If a requirement structure contains a requirement that has not got the status of 'Agreed', the requirements structure cannot get the status "Agreed". In other words, the following constraint ensures that only such requirements structures can get the status 'Agreed' which contain only 'Agreed' requirements objects. In Telos, this constraint looks as follows:

```

Individual RequirementsStructureConstraint with
constraint,attribute
  StatusOfAgreement_con : $forall f1/RequirementsStructure
f2/Requirement (f1 contains f2) and not(f2 StatusOfAgreement
Agreed) => not(f1 StatusOfAgreement Agreed)$
comment,attribute
  comment1: "This constraint ensures that only such
requirements structures can get status 'Agreed' that contain
ONLY agreed requirements object."
end

```

Figure 7: Constraint

As soon as one tries to change the status of a requirements structure to 'Agreed', the constraint triggers and checks whether there is any requirement contained in the structure without the status 'Agreed'. If so, a constraint violation has occurred and the changing of the status of the structure to 'Agreed' will be denied.

Active guidance, however, has the ability of advising the user what should be done next. There are three ways that active guidance has been implemented in RATS using Telos. The first way is to assign text strings to a class,

explaining what steps usually need to be performed with the instances of this class. The following frame shows an example of this kind of guidance:

```
Individual Goal in DEVmyc isA RequirementsCategory with
Description,attribute
  Description1 : "This goal should be refined in sub-goals,
implementation constraints or functional behaviour."
end
```

Figure 8: Active Guidance

The second way of implementing active guidance is to assign intelligence objects to specific instances of any class. An intelligence object is an instance of a meta-class called *Intelligence* and contains the necessary information to further develop the instance (e.g. a specific requirement), depending on its current status. There are two kinds of intelligence objects implemented in RATS: Object-related-intelligence and methodology-related-intelligence. Object-related guidance is concerned with conceptual issues on a lower level, while methodology-related guidance deals with methodological issues on a higher layer. Instances of the first type give advice on what should be done for this very specific object, in order to make it acceptable. Following is an example of an object-related intelligence object:

```
Individual Int28 in ObjectRelatedIntelligence with
ToDo,attribute
  ToDo1 : "This requirements object doesn't have a description.
In order to be complete you need to add a description to it."
end
```

Figure 9: Object-Related Intelligence

Instances of the methodology-related guidance help the designer through the development process of the whole service. The following frame gives an example of an intelligence object that provides methodology-related guidance:

```
Individual Int1004 in MethodologyRelatedIntelligence with
ToDo,attribute
  ToDo1 : "Define with the customer the service using the
service definition template. Address all headings contained in the
service definition template!"
end
```

Figure 10: Methodology-Related Intelligence

More specific active guidance is provided by parameter rules and parameter queries. Intelligence objects can have parameter queries associated with them. These queries contain complex rules that enable them to ask for specific parameters. This gives even more details on what needs to be done with certain objects.

5. CONCLUSION AND DISCUSSIONS

This paper has briefly presented three different kinds of languages, all suitable for the creation of models of systems and their environments. The motivation for the creation of these models is to develop better systems. Informal, semi-formal and formal languages have different strengths and weaknesses, and an approach has been described that allows the integration of these three languages, leveraging the advantages of each one of them. The knowledge-representation language Telos has been used to implement the suggested integration of conceptual models. The domain chosen was the telecommunications domain.

Conceptual models have the potential of aiding developers during the development process. Models can be analyzed, discussed and investigated for completeness and consistency. The challenge is to create models in such a way that the greatest benefit is achieved. This requires a lot more work. How much and what kinds of intelligence can be implemented using conceptual models? What is the best tradeoff between generic models that can be used for various domains and domain-specific models that require major changes if used in different settings?

The research described here will continue with completing an initial prototype of the RATS tool in order to do a more extensive case study. This will show weaknesses in the current approach and help refine the models.

6. REFERENCES

- [1] Chaos, Software Development Report by the Standish Group, <http://www.standishgroup.com/> - 1995
- [2] Sommerville, I., *Software Engineering*, Addison-Wesley, 1995.
- [3] Sommerville, I., and Sawyer, P., *Requirements Engineering: A Good Practice Guide.*, John Wiley & Sons, Chichester, England, 1997.
- [4] Loucopoulos, P. and Zicari, R., *Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development*, John Wiley & Sons, 1992.
- [5] ConceptBase home page, <http://www-i5.informatik.rwth-aachen.de/CBdoc/>
- [6] Mylopoulos, J., Borgida, A., Jarke, M. and Koubarakis, M., *Telos: A Language for Representing Knowledge about Information Systems*, ACM Transactions on Information Systems, 8, (4), pp. 325-362, 1990.
- [7] Eberlein A., Requirements Acquisition and Specification for Telecommunication Services, PhD Thesis, University of Wales, Swansea, UK, 1997
- [8] Chen, P.P.S., *The Entity-Relationship Model – Toward a Unified View of Data*, ACM Transactions on Database Systems, 1, (1), pp. 9-36, 1976.