ELECTRONIC **WORKSHOPS IN COMPUTING**

Series edited by Professor C.J. van Rijsbergen

B Novikov, University of St Petersburg, Russia and J W Schmidt, University of Hamburg, Germany. (Eds)

Advances in Databases and Information Systems, Moscow 1996

Proceedings of the International Workshop on Advances in Databases and Information Systems (ADBIS '96). Moscow, 10-13September 1996

Interfacing of Object Analysis and Design Methods with the Method for Interoperable Information Systems Design

Dmitry O. Briukhov

Published in collaboration with the British Computer Society

BCS

Springer

Interfacing of Object Analysis and Design Methods with the Method for Interoperable Information Systems Design

Dmitry O. Briukhov
Institute for Problems of Informatics
Russian Academy of Sciences
Moscow, Russia
E-mail: brd@ipi.ac.ru

Abstract

In our days numerous object analysis and design methods appeared. These methods provide users with easy to understand graphical notations for expressing a wide variety of concepts central to the presentation of software requirements. While such techniques are recognized as useful tools, the graphical notations used with these methods are often ambiguous, resulting in diagrams that are easily misinterpreted. The disadvantage of these methods also is a weak support of reusability.

On the other hand, the Synthesis method being developed [3, 1] is focused on the interoperable information system design basing on the reuse the pre-existing information resources (databases, legacy systems, program packages, etc.). The Synthesis method is a top-down, bottom-up iterative process of analysis, design and development. To take an advantage of the existing Object Analysis and Design OAD methods we embed the Synthesis method into one of such methods so that the phases of requirement planning and domain analysis could be developed using the chosen method technique. After that the Synthesis method continues a design with the reuse of pre-existing resources.

Therefore a problem of interfacing of an OAD method and the Synthesis method arises. In this paper we present the mapping of the graphical notation of a specific OAD method (OMT) to the Synthesis entities. In order to prevent ambiguities in the graphical notation in OMT we impose some restrictions on OMT using. Also we augment OMT with the ontological specifications needed to resolve contextual differences between application and pre-existing information resources as well as with predicative specifications of object behaviors.

1 Introduction

The results reported in the paper were obtained in frame of the Synthesis project[1] that is focused on various problems of Heterogeneous Interoperable Information Resource Environment (HIRE) specification, design and management. Synthesis mainly attempts *semantic interoperation* issues intended for the specification and design of interoperable information systems [4].

In the project the Synthesis method for the interoperable information system design is being developed [1]. The method is based on reuse of pre-existing information resources (databases, legacy systems, program packages, etc.). The method emphasizes the design of semantically interoperable compositions of the pre-existing information resources in HIRE. Some specific architecture supporting HIRE (such as [8]) is assumed. Resources should be semantically coherent and their compositions should be consistent and meaningful within the application context. The Synthesis method focuses on the semantic interoperation reasoning process that should lead to concretization of specifications of requirements by views over the pre-existing information resources [4].

The Synthesis method is a top-down, bottom-up iterative process of analysis, design and development. The schema sketching the method is shown on the figure 1.

In the method we focus on the design phase and for requirement planning and domain analysis phases we use conventional techniques of object analysis and design. The architectural framework for Synthesis is designed so that

[1] This research partially was supported by the INTAS Program grant INTAS-94-1817

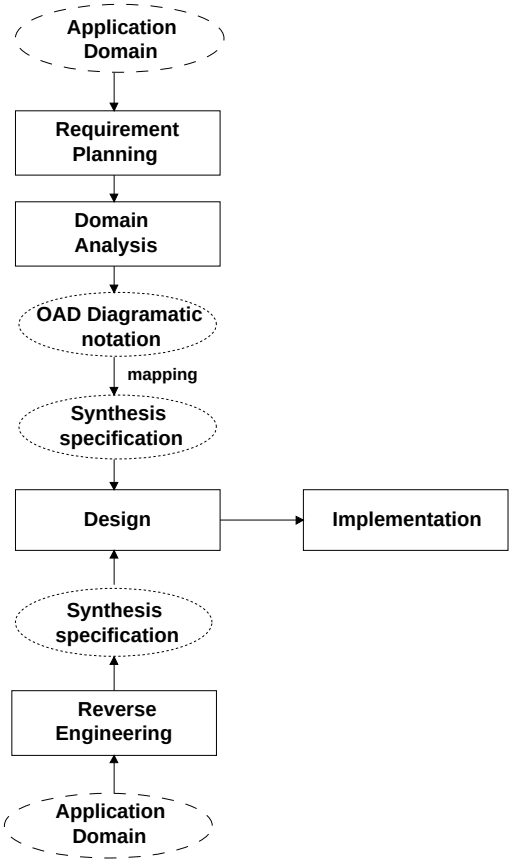

Figure 1: Synthesis method sketch

the Synthesis method itself should be neutral to existing (OAD) methods [7]. We take OMT as one of the most powerful and popular OAD methods.

The results of these initial phases (OMT object diagrams) are interpreted as the Synthesis specifications of the application requirements.

To work with reuse effectively, OAD methods should provide facilities to search for suitable components and check their conformance to requirements. For these purposes we augment OMT with the ontological specifications.

Synthesis language uses a symbolic notation having precise semantics that is required to reason that certain resource is reusable for a given specification. Mapping of a model developed by a conventional OAD method into a Synthesis model gives to this model precise semantics. Thus Synthesis language can play a role of a metamodel [10] with respect to a conventional OAD method.

The paper is organized as follows. Section 2 provides an introduction to OMT and object model notation. Section 3 gives a short summary of the main Synthesis concepts. Section 4 presents the mapping of OMT object model notation to the Synthesis entities. Section 5 provides an augmentation of OMT with the ontological specifications.

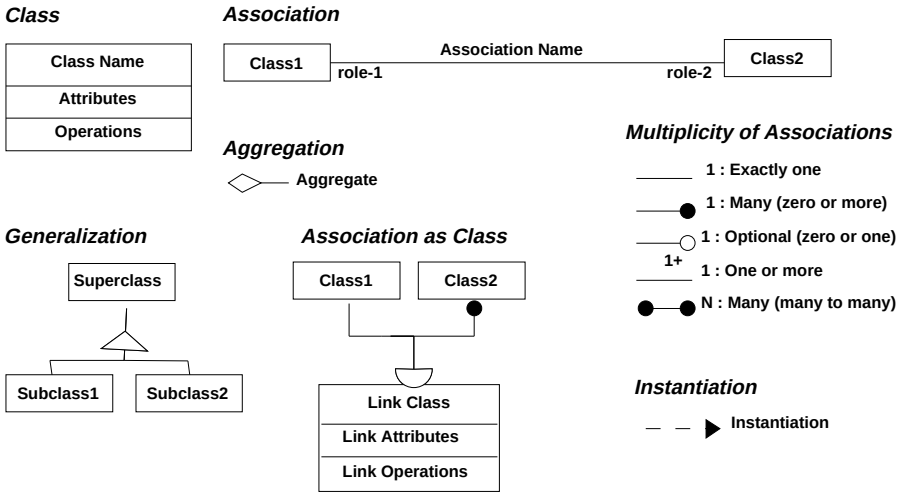

Figure 2: The basic OMT object model notation

2 A Short Introduction to OMT

2.1 The OMT Methodology

The Object Modeling Techniques[9] provides a comprehensive coverage of the object modeling. It includes the analysis modeling, design modeling, and implementation modeling. The purpose of analysis is to model the real world so that it can be understood. The system design stage concentrates exclusively on the overall architecture of the system. The object design stage concentrates on optimizing and refining the object model, ready for translation into a programming language. Three models of the system are developed initially and then refined in all these stages: the object model, the dynamic model, and the functional model. The object model represents the static, structural, ”data” aspects of a system. The dynamic model represents the temporal, behavioral, ”control” aspects of a system. The functional model represents the transformational, ”function” aspects of a system.

2.2 The Object Model Notation

The Object Model concepts include objects, classes, attributes, operations, generalization relationship, instantiation relationship, associations, aggregation relationships, qualifiers, abstract classes, class attributes and operations, link attributes and operations, links as classes and modules.

Figure 2 shows the basic OMT object model notation.

Links and *associations* establish relationships among objects and classes. A link connects two or more objects. An association describes a group of links with common structure and common semantics. *Multiplicity* specifies how many instances of one class may relate to each instance of another class. Associations are inherently bi-directional. A *role* is a direction across an association.

The method allows to define attributes of a link. The *link attribute* is a named data value held by each link in an association. Sometimes it is useful to model an *association as class*. Each link becomes one instance of the class.

The method allows to define a qualified association that relates two classes and a qualifier. A *qualifier* is a special attribute that reduces the effective multiplicity of an association. The qualifier distinguishes among the set of objects at the many end of an association.

Aggregation is the ”part-whole” or ”a-part-of” relationship in which objects representing the components of something are associated with an object representing the entire assembly. It is a tightly coupled form of association with

```
{<class name>;                              {<type name>;
 in: class, <list of metaclasses>;           in: type, <list of metatypes>;
 superclass: <list of superclasses>;         superclass: <list of supertypes>;
 params:{<list of formal parameters>};       params:{<list of formal parameters>};
 class_section: {<type>}                     <list of attributes and functions>
 instance_section: {<type>}                 }
 instance_instance_section: {<type>}
}
```

Figure 3: An example of the Synthesis specification

some extra semantics.

A *class attribute* describes a value common to an entire class of objects. A *class operation* is an operation on the class itself. In method they described as attributes and operations with prefix $.

A *module* is a logical construct for grouping classes, associations, and generalizations. Modules enable to partition an object model into manageable pieces. The same class may be referenced in different modules. In fact, referencing the same class in multiple modules is the mechanism for binding modules together.

3 The Basic Synthesis Entities

The Synthesis [5] entities include objects, classes, types, generalization/specialization type hierarchies, subclass hierarchies, classification hierarchies made by metatype relationship, specification inheritance, attributes, functions defined by predicative specifications as object calculus definitions, assertions related to values and their attributes, association metaclasses, modules and ontological specifications.

Abstract data types constitute the basis for the type system of the language providing for construction of arbitrary data types. Synthesis supports the multilevel type system that sets a classification relationship on the data types. A type in the model is a value represented by an object. A type as the value may be produced as the result of type expression evaluation used mainly for type inferencing. A class supports a set of objects of a given type that constitutes an extent of the class. Two types are associated with a class. One defines an interface of the class and other defines an interface of object instances of the class. With metaclasses type that defines an interface of the instances of instances of the metaclass is additionally associated.

Object attributes are treated also as classes of association objects establishing a correspondence between a set of objects in an association domain and a set of objects (values) in an association range. Thus a specification of an attribute is considered as a specification of an association class. Treating of an attribute as an association class provides for introduction of association metaclasses setting a classification relationship on attribute classes and establishing properties of association classes and/or their instances.

The Synthesis entities are represented by frames. Figure 3 shows as example the specifications of classes and types.

4 Mapping the OMT Object Model Notions into the Synthesis Entities

Classes Generally a class of OMT is mapped into a Synthesis class, its own type and a type describing an interface of instances of the class, except the cases which will be described later. In such cases we treat OMT classes as having persistent extensions.

For OMT classes that according to the application requirements should not provide such extensions we introduced special tags as specific descriptions in OMT notation. In such cases we map classes of OMT to types in Synthesis.

Figure 4 shows an example of the mapping for class *proposal*.

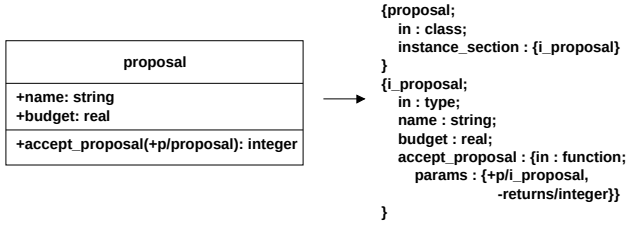

Figure 4: An example of mapping of the OMT class

Attributes and Operations Each operation should be defined by predicative specifications using object calculus definitions.

Attributes and operations of an OMT class are mapped to the attributes and functions of a type describing the interface of instances of the class.

Figure 4 shows an example of the mapping for attributes *name, budget* and operation *accept_proposal* of class *proposal*.

Associations We restrict ourselves only with binary associations. Each association may not have a name, but should have the role names (at least one).

An association is mapped to the attribute (or attributes, if two role names are defined). The attribute becomes an instance of an association metaclass reflecting semantics of the OMT association. The attribute name is the name of the role of the association. The attribute type is a type corresponding to opposite class for one-to-one association or a template set defined on a type corresponding to opposite class for one-to-many association. Many-to-many association is considered as two one-to-many associations.

Figure 5 shows an example of the mapping the association relationship between classes *proposal* and *person*, *proposal* and *researcher*.

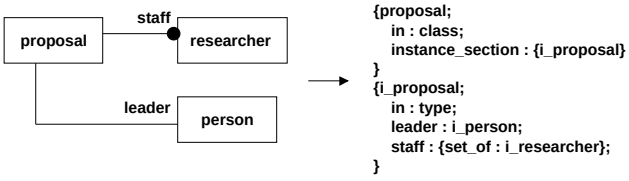

Figure 5: An example of mapping of the OMT associations

Generalization relationship Generalization relationship is mapped as the class generalization and type subtyping relationships in Synthesis described by the slot $super$ in a class and type specifications.

Figure 6 shows an example of the mapping the generalization relationship between classes *person* and *researcher*.

Instantiation relationship Instantiation relationship is mapped as the classification relationship in Synthesis and described by the slot in in a class and type specifications.

Each OMT class having as instances other classes should be considered as metaclass in Synthesis. Such class is mapped to the metaclass in Synthesis.

Figure 7 shows an example of the mapping the instantiation relationship between classes *meta_proposal* and *proposal*.

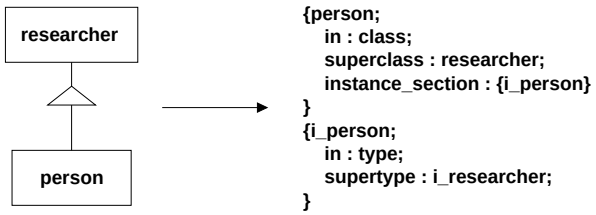

Figure 6: An example of mapping of the generalization relationship

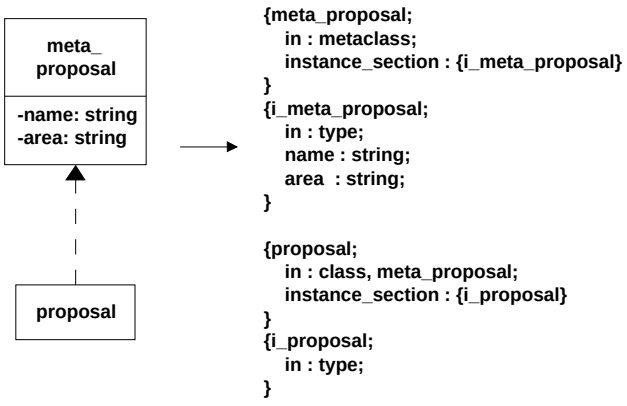

Figure 7: An example of mapping of the instantiation relationship

Class Attributes and Class Operations Each operation should be defined by predicative specifications using object calculus definitions.

Class attributes and operations of a class are mapped to the attributes and functions of the own type describing the interface of the class as an object.

Figure 8 shows an example of the mapping for class operation *new* of class *proposal*.

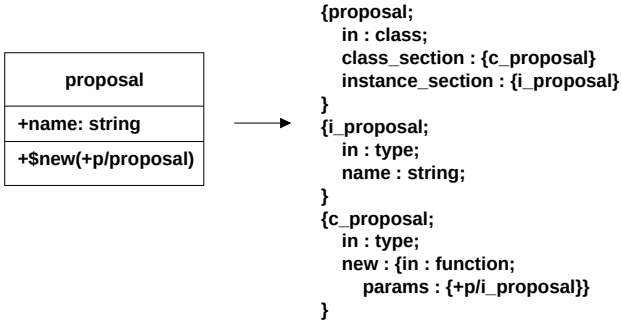

Figure 8: An example of mapping of the class attributes and operations

Aggregation relationships The aggregation relationships are mapped similarly to the associations.

Association as class Such classes are mapped to the association metaclasses of Synthesis. Classes having these classes as instances are also mapped to the associations metaclasses.

Figure 9 shows an example of the mapping the association as class between *budget* association and *budget_sem* class.

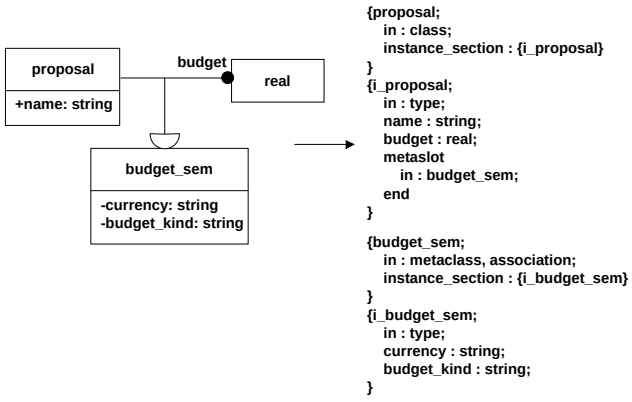

Figure 9: An example of mapping of the OMT associations as classes

Modules OMT modules correspond to the Synthesis modules.

5 Augmenting OMT with the Ontological Concepts

5.1 Synthesis Ontological Specifications

The specifications resulting from the requirement planning and domain analysis phases should be complete for the semantic interoperation reasoning. For this purpose we augment OMT with the ontological specifications needed to resolve contextual differences between application and pre-existing information resources.

An *ontological specification* is a set of definitions of context-specific knowledge representation primitives for describing of context vocabulary (names of individuals, functions, predicates, attributes, types, classes) in a form that is both human and machine readable. Ontology contains also a set of rules (axioms) associated with the vocabulary or with particular terms [6]. Ontologies provide a conceptual framework for talking about an application domain and an implementation framework for problem solving. Ontologies play the role of a coupling interfaces among shared resources providing the basis for them to interoperate.

In Synthesis ontological specifications usually are organized in a separate ontological module coupled with the information resource specification module. The ontological module consists of a collection of *name definitions* and *concept definitions.*

A *name definition* may contain verbal definitions (human and machine readable definitions of concepts in the natural language) and interconcept relationships (such as hypernym/hyponym, positive, negative). Name definitions are defined as objects of a class *OntDef*.

Concept definitions syntactically look as class (metaclass) definitions containing a metainformation associated with the concept (e.g., attributes, set of rules).

Figure 10 shows the Synthesis example of information resource specification module and ontological module for the class *proposal* with attribute *budget* and association metaclass *budget_sem.*

5.2 Representation of Ontological Specifications in the OMT Graphical Notation

We augment OMT with ontological analysis. The results of the analysis are represented in the OMT ontological modules having the following form. Name definitions are represented as objects of OMT class *OntDef*. The name of each such object is the name of the corresponding concept given with prefix ”O”. The hypernym/hyponym, positive and negative relationships between concepts are represented by links between objects with corresponding names.

```
Information resource specification module:

{Proposal;                                    {proposal;
  in: type;                                     in: class;
  . . .                                         instance_section: {Proposal}
  budget: real;                               }
  metaslot
    in: budget_sem;
  end
}


Ontological module:

{proposal;                                    {budget_sem;
  in: OntDef;                                   in: metaclass, association;
  def: {a research and development              instance_section: {Budget_sem}
    activity of a consortium that may         }
    be in a state of review, acceptance,
    rejection, termination} }                 {Budget_sem;
                                                in:type;
{budget;                                        budget_kind: string;
  in: OntDef;                                   currency: string
  def: {the quantity of money                   }
    limiting the spending}                    }
}
```

Figure 10: An example of the Synthesis ontological specification

Concepts represented as classes, metaclasses and association metaclasses in Synthesis are pictured as classes in the OMT graphical notation with predefined attribute (*in_class*, *in_metaclass* and *in_association* correspondingly).

Relationships between a class and associated types (defining an own interface of the class and an interface of instances of the class) are represented in OMT by associations between class and types (named *class_section* and *inst_section* correspondingly). For metaclasses the *inst_inst_section* association is added.

Figure 11 shows an example of representation of ontological specification for class *proposal* and attribute *budget* and of association metaclass *budget_sem* in the OMT object model notation.

6 Conclusion

In this paper we define the interfacing of the OAD method (OMT) to the Synthesis method intended for a transition from conventional OAD method to the interoperable information system design. In the Synthesis method we focus on the design with reuse and apply OMT to cover the requirement planning and domain analysis phases. The mapping of the OMT object model notation into the Synthesis entities is given.

To achieve the required completeness of the specifications for the semantic interoperation reasoning we augment the OMT with the ontological specifications as well as with the predicative specifications of functions.

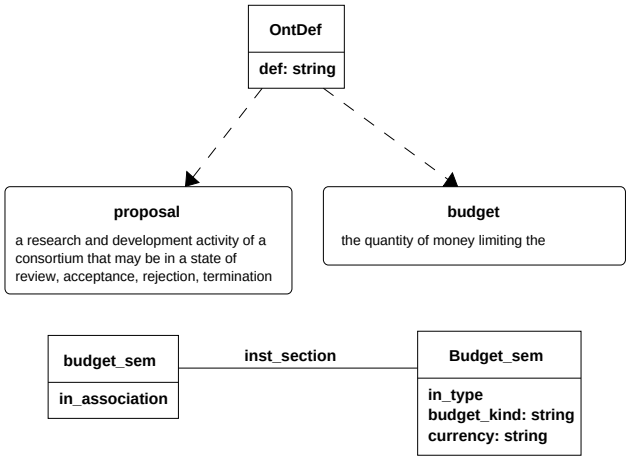

Figure 11: An example of representation of ontological specifications in OMT

References

[1] K. Berg, L. Kalinichenko, Modeling Facilities for the Component-based Software Development Method. Submitted to the International Workshop "Advances in Databases and Information Systems", Moscow, 1996

[2] D. O. Briukhov, S. S. Shumilov, Ontology Specification and Integration Facilities in a Semantic Interoperation Framework. In *Proceedings of the 2nd International Workshop on Advances in Databases and Information Systems*, Springer, 1995.

[3] Kalinichenko L. A. Emerging semantic-based interoperable information system technology. Computers as our better partners. Proceedings of the International IISF/ACM Symposium, Tokyo, World Scientific, 1994

[4] Kalinichenko L. A., Constructing of Concretization of an Application Domain Description by Information Resources., Advances in Database Systems. USIM, N. 6, 1994, Kiev.

[5] Kalinichenko L. A., SYNTHESIS: a language for description, design and programming of interoperable information resource environment. Institute for Problems of Informatics of the Russian Academy of Sciences, (in Russian), September 1993.

[6] Kalinichenko L. A. Rule-based concept definitions intended for reconciliation of semantic conflicts in the interoperable information systems. In *Proceedings of the Second International Baltic Workshop on DB and IS*, Tallinn, 1996 (to appear)

[7] A. Hutt (editor), Object Analysis and Design. Description of methods. Object management group. John Wiley and Sons. 202p., 1994.

[8] Object Management Group, "The Common Object Request Broker: Architecture and Specification", OMG Document Number 91.12.1, December 1991.

[9] J. Rumbaugh et al., Object-Oriented Modeling and Design, Prentice Hall, 1991.

[10] OMG Object Analysis and Design RFP-1, OMG TC Document ad/96-05-01, 1996.