Using a Data-Oriented Approach to Decide on Similarity of Objects During Domain Analysis

Johann Eder

Institut für Informatik, Universität Klagenfurt,
Universitätsstr. 65, A-9020 Klagenfurt, Austria

and

Wilhelm Rossak

Institute for Integrated Systems Research,
Dept. of Computer and Information Science,
New Jersey Institute of Technology, Newark, NJ 07102, USA

Abstract

Domain analysis has been recognized to be an important factor in systems engineering. However, due to the relatively short time we have spent investigating this concept of a conceptual model on a level above the single project, we still face severe problems with respect to methodologies and tool-support. Our contribution focuses on the problem to decide on similarities between (data) objects. We suggest the use of a meta (data) structure which handles similarity as a semantic concept. This model goes beyond the usual classification and generalization techniques by supporting so called "generic properties". Using such a meta model, we will be able to define an algorithm which retrieves from the meta data structure a set of objects which are similar to a user-defined starting set of object descriptions. This approach will provide us with a basis to integrate objects of an application domain into one comprehensive structure and to approach the problem of multiple overlapping domain models.

1 Introduction

System engineering strives to go beyond software engineering in handling the development of systems with a holistic approach. The key issue is to recognize, to analyze, and to solve the problems inherent in system development, problems which go beyond a single application and beyond hardware and software engineering [6]. Based on this general concept, domain analysis has become a major issue and a field of research in its own right [8], [1].

Domain analysis has the goal to result in a comprehensive and integrated domain model that describes entities, functionality, data structures, relationships and dependencies of a part of the real world that has been identified as an application domain. All the objects in this model are related to one type of application, e.g.

banking, book-keeping, corporate information systems, etc., and constitute the application domain model. This application domain model is not targeted towards one single project. It is the basis and the reference frame for the requirements analysis and the further development of all projects in the domain of applications it describes.

However, the concepts and methodologies of domain analysis are quite young and quite immature as compared to other techniques. Fundamental questions - What is a good size for an application domain? How do I determine the boundary of my domain? How can I integrate two domain models? How can I compare domain models? - have not been answered in a satisfactory manner and hinder any widespread use. We see two major reasons for these problems:

- Domain models have to deal with the semantics of the objects they describe.
- Domain models will have overlapping application areas, thus possibly duplicating application domain information in a redundant way and/or describing it from different points of view.

As an example, book-keeping is a part of banking and may be also part of a corporate information system. Therefore, it would be good practice to be able to use one domain model for book-keeping and to integrate and incorporate it in the other two, superordinate domain models. However, this makes it necessary to decompose and to aggregate domain models and to compare and classify the objects of these models. Such a classification and comparison must go beyond the limits of syntax analysis and take into account the semantics of the involved objects.

The basic question in all these activities is to express similarity and differences between objects of one or more domain(s). Based on a similarity analysis, integration techniques can be used to specify objects unambigously and to derive a consistent and integrated model of the application domain.

Section 2 gives a short overview of domain analysis as used in systems integration and highlights the differencies to existing approaches. Section 3 discusses the structure of the meta data model and the retrieval algorithm for similar objects. We finally conclude with a summary of the proposed concepts.

2 Domain Analysis for Systems Integration

Systems integration has been seen for a long time as a bottom-up, post-facto approach, to glue together already existing pieces of a system [10]. However, if we take the point of view of a systems engineer, as discussed above, systems integration should be a vital and preplanned part of the engineering process.

Such a holistic, system oriented approach derives three major results, which constitute also an outline of the sequence of tasks to be applied [11]:

- A domain model.
- An integration architecture.

- The specification of enabling technologies.

The application domain model is the basis for all future decisions and the first result of the engineering process. It is a product of a requirements engineering process. However, it does not focus on the needs and requirements of one single application. It derives a comprehensive model of the application domain and is independent from any specific project effort [9].

The integration architecture is the meta design for systems in the application domain. Once again, we do not want to look at specific projects but want to provide a generic framework of design decisions and guidelines. Such an architecture can be detailed enough to predefine the handling of global data, communication between processes, decomposition guidelines, etc., but also generic enough to leave sufficient freedom for the flexible development of applications in the domain [10], [11]. The domain model is one of the major factors in deciding on a suitable integration architecture [11].

Enabling technologies constitute the implementation framework for integration architectures by providing the means to realize ongoing and future projects. These technologies have to be "fine-tuned" to support the design decisions of the integration architecture, to support the needs of application domain, and to provide flexibility for projects in the domain [10].

In this paper, we want to focus on domain analysis as the very basis for all other integration efforts. This means that a domain model serves as a specification of the "part of the world" where a set of systems will be (or already has been) developed. Therefore, it will be - together with the integration architecture as the "technical guideline" - the basis and starting point for the requirements phases of projects in this domain, providing a framework of concepts and objects for this project phase.

A typical approach to dertive project requirements in the presence of a domain model is to link new objects and functions, as necessary for the project, to similar already existing objects of the domain model, thus preserving a consistent and integrated view of the application. This task makes it necessary to classify objects according to their (semantic) similarity and to search similar objects in the domain model.

Some work has been done to support domain analysis based on a variety of concepts. An overview and additional reading on this topic can be found in [8] and in [1]. Most of the proposed solutions and concepts are oriented towards the functionality of the domain and have the goal to derive later on software structures in a reuse oriented environment. The DRACO project [7], coining the term domain analysis, can be seen as the ancestor of most of these reuse centered approaches.

We want to discuss the data and integration aspects of domain and similarity analysis. Our goal is to complement functionally oriented approaches and to provide support for the integration and aggregation/decomposition of domain models from the point of view of a system engineer. We go beyond the scope of other projects, e.g. [5], by abstracting from particular notations and by providing a generalized framework for classification and similarity analysis. Similarity is seen as a semantic concept, a point of view which is supported by the introduction of generic properties to describe and to classify (data-)objects of the application domain on basis of their semantics.

3 A Meta Data Model Based on Similarity

3.1 *Structure of the Meta Data Model*

The meta data structure we discuss is organized to contain the conceptual data models of different applications or projects in the same or in overlapping application domains. The key idea for the proposed structure is to abstract from the use of a specific modeling technique and to provide a general lattice for representing conceptual models and semantic objects [4]. This general framework can be used with various semantic data models like the (extended) Entity-Relationship model [2], or RM/T [3]. However, we assume that for a particular meta data structure all included conceptual models use the same modeling technique. Therefore, we do not consider the problems of heterogeneous data models and their mappings.

The central building block of the meta data structure is the conceptual model representing the data domain of a specific application. A conceptual model consists of semantic objects and the relations between them. So we consider all items of a semantic model (like entities, relationships, attributes, value-domains) as semantic objects. This viewpoint has the advantage of great flexibility while maintaining simplicity of structure.

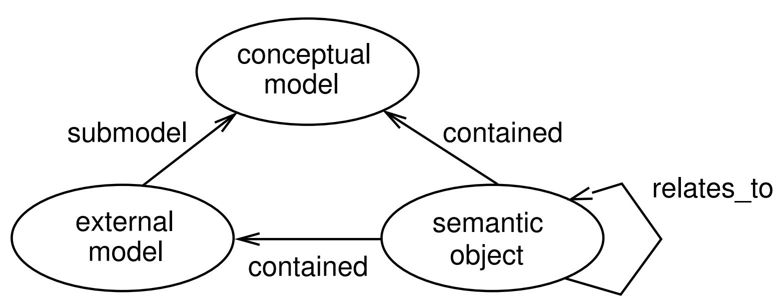

Fig. 1: Structure of the meta data model

Furthermore our meta structure contains external models representing the view of of a certain part of an application on the conceptual model. An external model is more than just a part of a conceptual model. It contains it's own set of semantic objects which are derived in a mapping process from the conceptual model.

Obviously there may be various external models for a certain conceptual model. On the other hand, a specific external model can be derived from different conceptual models by different mappings. However, we do not consider external models spanning several conceptual models (i.e. being derived from different conceptual models at the same time by a single mapping, like in heterogeneous data models [12]) .

In the meta data structure we describe which semantic objects are contained in (a) domain model(s) - be it a conceptual or external model. Furthermore, we represent how the semantic objects are related through the relationship "relates-to". This relationship contains the type of the relation, the involved semantic objects, and the role these objects have in this relationship. The type of the relationship can be aggregation, association, generalization and cover aggregation.

To give an example, we consider a simple E-R model with the entities employee (with the attributes name, number and birthdate) and department (with the attributes code and location) connected by the relationship works-in. In our meta model all entities, attributes, and relationships are considered to be semantic objects. Through "relates-to" the connections between these objects are represented, i.e. that employee relates-to name, number and birthdate.

An advantage of this treatment of different concepts in form of the unifying notion of semantic objects is that we can find similarities also for ontologically different structures. To give an example: In one application spouse ofan employee may be modeled as an entity linked to the employee entity by a relationship "is-spouse-of". In a different application spouse may be an attribute (property) of an employee object. Nevertheless, there is a similarity between the two models which probably is interesting for a system engineer.

This structure is general enough to represent various types of semantic relationships. The "relates-to" construct can be seen as representing the edges of an E-R diagram, or a semantic network, or the graph relations of RM/T, etc..

We construct this meta data model in order to reason about similarities between different models [4]. Currently, connections between different models can only be found if they contain identical semantic objects. For our purpose it is important to address commonalities between different models beyond identity. Therefore, we would like to have a 'similarity relationship' defined between different models. For this reason we define a generalization hierarchy for semantic objects. We consider two objects as being similar, if they have a common ancestor in the generalization hierarchy. To adapt to different viewpoints and to different abstractions, we allow a semantic object to be a specialization of several generic objects (multiple inheritance).

Now we can regard two conceptual models to be similar, if the objects they contain are similar with respect to the generalization hierarchy. We do not express similarity between models explicitly. The degree of similarity between models depends on one hand on the number of objects being similar, and on the other hand on the degree of similarity of the respective objects. By the set of objects of one model having similar counterparts in the other model we can find similar submodels (intersection of models based on similarity).

A unique aspect of our approach is the treatment of properties of semantic objects as objects themselves. This allows us to create a generalization hierarchy as indicator for similarity even for properties. This concept leads to the notion of generic properties.

To give an example for generic properties, we assume a company to have different kinds of employees with the following properties:

```
manager:     number, name, grade
salesperson: number, name, sales, district
programmer:  number, name, lines-of-code, languages
```

Managers, salespersons, and programmers have employee numbers and names. These properties are common to all three objects and are identical in identifier, type, etc. Therefore, according to usual classification procedures, they are also primary candidates to be properties of a more generic object 'employee'.

The properties grade, sales, and lines-of-code have no syntactical commonality. However, on a semantic level, they can be interpreted as indicators for the success of the respective class of employees. We thus can generalize these properties to a generic property 'success-indicator'. This generic property is associated with the employee object to model the three possible different forms of indicators found for managers, salespersons, and programmers:

```
employee: number, name, success-indicator
```

In most object oriented systems and classification structures, each subordinate object in a generalization hierarchy inherits all properties from the superordinate object in a syntactically guided procedure, leading to identical identifiers, types, etc.. In our meta structure, a subordinate object may substitute an inherited property with a (semantically) similar one. Therefore, objects in higher levels of the generalization hierarchy can usually be associated with more (generic) properties than in a pure inheritance lattice. This means that in our model objects on higher levels can be associated with more semantic content. It also means that we extend the definition of similarity in our structure into semantic categories.

3.2 *Search for Similar Objects*

The goal of constructing this meta data structure is to assist a system engineer during domain analysis. It can be used for describing the objects of an application domain, and especially for searching for similar objects which have already be analyzed, as well as for deriving parts of different application domains which can be considered as similar. To do this job, he has to classify new objects and to search for existing (similar) ones.

Both alternatives require the definition of a proper search algorithm. As manual browsing through all models and objects is not feasible, we provide a search strategy to determine a subset of models and objects which are more likely to be useful. For a more detailed description and an application see [4]. In this paper, we confine ourselves to a schematic overview of the relatively complex search process.

The search starts with a given set of objects and looks for similar objects by using links in the generalization hierarchy. Furthermore it considers the properties of the given objects (which are objects themselves) and determines similar properties and objects, also through the generalization hierarchy.

The general ides of the search process is a users driven search through the generalization hierarchies. The system presents a set of possibly relevant semantic objects to the user. As similarity is a semantic concept which is also influenced by the viewpoint of the system enginner, the ultimate decision about similarity has always be made by the system engineer.

From the given set the system starts to look for similar objects which are objects that have comon ancestors in the generalization hierarchy with objects from the given set, or which have properties which are similar due to this hierarchy.

If the set of objects found is this process is too large, the search continues on a higher level of the generalization hierarchy, where the amount of objects is usually much smaller. This set are the common ancestors of the found objects. This set of higher level objects is then presented to the system engineer who chooses those which he considers as beeing relevant for his project.

From this set of relevant higher level objects the search then goes on downwards by presenting descendants of these objects to the system engineer, who again chooses, and so on. If the set is too small or the presented objects become irrelevant, the direction of the search can be reversed, and backs up to higher level objects. The system engineer can direct the search by specifying similarity measures, e.g. the length of the path between two objects in the generalization hierarchy.

Generic properties are useful in two ways in this search. First, the search for similar objects can use the generalization hierarchy for properties to determine similar properties and look for objects which are associated with these properties. Second, as pointed out above, higher level objects may be associated with generic properties, thus giving more information than usual on the semantics of objects on a higher level. This information can be used by the system engineer in the selection of relevant objects on higher levels. Therefore, he will make better decisions which objects are relevant and will be able to avoid browsing through irrelevant branches of the hierarchy.

4 Summary and Conclusions

From the point of view of the system engineer, a meta data structure, as described above, provides a framework to classify and to compare (data-)objects of an application domain (or objects of different application domains) based on their (semantic) similarity. This is a first step to support aggregation/decomposition and integration of application domains.

By defining the relationships between data and functionality, e.g. [4], a link to a functionally oriented description of the domain can be established and used to classify functions as "similar", based on their use of similar data. The application of such a link and the integration of the approach described in this paper in a complete domain analysis/ systems integration methodology are part of ongoing research efforts.

References

[1] *Participant's Proceedings of the Domain Modeling Workshop at the 13th International Conference on Software Engineering*, Austin TX, USA, May 1991.

[2] P. P. Chen. The entity-relationship model: Towards a unified view of data. *ACM TODS*, 1(1):9 –36, 1976.

[3] E. F. Codd. Extending the database relational model to capture more meaning. *ACM TODS*, 4(4):397 – 434, 1979.

[4] J. Eder and W. Rossak. An archive of semantic data models as basis for software reuse. In *Proc. 8th International Symposium on Applied Informatics*, pages 284–287, Innsbruck, Austria, February 1990.

[5] C. Hsu, M. Bouziane, W. Cheung, J. Nogues, L. Rattner, and L. Yee. A metadata system for information modeling and integration. In *1st International Conference on Systems Integration*, pages 616–624, Morristown NJ, USA, April 1990. IEEE Computer Society Press.

[6] IEEE. *Minutes of the Computer Based Systems Engineering (CBSE) Taskforce Meeting*, 13th ICSE, Austin TX, May 1991.

[7] J.M. Neighbours. The DRACO approach to constructing software from reusable components. *IEEE TR-SE*, 10(5):564–574, September 1984.

[8] R. Prieto-Diaz and G. Arango. *Domain Analysis and Software Systems Modeling.* IEEE Computer Society Press Tutorial. IEEE Computer Society Press, Los Alamitos CA, 1991.

[9] W. Rossak. Domain modeling and systems integration. In [1], pages 150–153.

[10] W. Rossak and P.A. Ng. Some thoughts on systems integration - a conceptual framework. *International Journal of Systems Integration*, 1(1):97–114, 1991.

[11] W. Rossak and S. Prasad. Integration architectures - a framework for systems integration decisions. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, Charlottesville VA, USA, October 1991.

[12] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, an autonomous databases. *ACM Computing Surveys*, 22(3):183 – 236, September 1990.