Ontology-Based Method Engineering

Ali Niknafs, Mohsen Asadi, Hassan Abolhassani

Sharif University of Technology, Computer Engineering Department, Tehran, Iran

Summary
The emerging field of semantic web technologies promises new stimulus for Method Engineering. However, since the underlying concepts of the semantic web have a long tradition in the knowledge engineering field, it is sometimes hard for method engineers to overlook the variety of ontology-enabled approaches to Method Engineering. In this paper we present an example of ontology application within the Method Engineering and its most popular approach assembly-based Method Engineering. Therefore, we propose an ontology-based approach to Method Engineering by adopting assembly-based approach and augmenting it with ontology concept.

Key words:
Ontology, Method Engineering, Assembly-based Method Engineering

1. Introduction

The word "ontology" comes from the Greek *ontos*, for "being", and *logos*, for "word". In philosophy, it refers to the subject of existence, i.e. the study of being as such. More precisely, it is the study of the categories of things that exist or may exist in some domain. A domain ontology explains the types of things in that domain. Informally, the ontology of a certain domain is about its terminology, all essential concepts in the domain, their classification, their taxonomy, their relations, and domain axioms. The ontology is an extremely important part of the knowledge about any domain. Moreover, the ontology is the fundamental part of the knowledge, and all other knowledge should rely on it and refer to it. There are many definitions of the concept of ontology in AI and in computing in general. The most widely cited one is "*Ontology is a specification of a conceptualization*" [1].
Ontologies provide a number of useful features for intelligent systems, as well as for knowledge representation in general and for the knowledge engineering process.

In this paper we provide a comprehensive description of the Method Engineering discipline, its tool support, and the most popular approach to Method Engineering, called assembly-based Method Engineering. Thereafter, we propose an ontology-based approach to Method Engineering by adopting assembly-based Method Engineering presented in [2]. In our approach we develop a semantic system to define the semantics of method fragments and perform the Method Engineering activities based on the ontology defined for the Method Base.

2. Method Engineering

The general conclusion derived from former researches in Method Engineering literature is that there is no one general-purpose information systems development method that can be applicable to all different situations. Thus, they will undoubtedly require project-specific methods and tools for supporting them. *Situational Method Engineering* (SME) aims at providing techniques and tools allowing to construct project-specific methods instead of looking for universally applicable ones.
Assembly-based Method Engineering which is the most popular approach to Method Engineering, is characterized by using reusable method portions, called *method fragments* or *method chunks*, which can be extracted from several existing methods [3]. A method fragment can be either a *product fragment* or a *process fragment*. A product fragment captures product related knowledge of methods whereas a process fragment captures activity related knowledge. Product fragments are deliverables, documents, models, diagrams or concepts. Process fragments are stages, activities and tasks to be carried out [4]. Computerized support of Method Engineering needs the method fragments to be stored in a repository called Method Base. In [2] three distinct approaches to SME are proposed. The assembly-based approach consists of three steps: specify method requirements, selecting method fragments, and assembling them. *Extension-based* SME aims at adapting and extending an existing method with new features. Whereas, in *paradigm-based* approach a new method is developed by instantiating, abstracting or adapting an existing meta-model.

A new method design approach composed of several SME approaches is proposed in [5]. This approach of method design uses the alternative method engineering approaches for different parts of the process and at different levels of abstraction. It also provides a

Manuscript received August 5, 2007
Manuscript revised August 20, 2007

framework allowing flexible application of four method development approaches namely:

- *Instantiation* approach: with the focus on instantiating an already available process meta-model;
- *Artefact-oriented* approach: devising a seamless complementary chain of artefacts and building the process around it;
- *Composition* approach: using one of the already available libraries of process patterns; and
- *Integration* approach: integrating features, ideas and techniques from existing methods.

Two of these approaches, Instantiation and Composition, are analogous to the Paradigm-based and Assembly-based approaches, respectively, while the Integration approach is particularly nonconformist in comparison to usual method engineering practices, in that it promotes integrating ideas and techniques directly from existing methods, instead of first dissecting the methods into method fragments and then store them in a method repository, as is common practice in assembly-based method engineering approach. The notion of this approach is that breaking down the methods into fragments may result in loss of functional capacity. Thus, Integration and Artefact-oriented approaches are relatively novel in this concept. Any of these approaches can be used for designing the method.

2.1. Assembly-Based Method Engineering

Assembly-based method engineering approach, also known as *Reuse technique* [3], is the simplest way to build new information systems development methods. In this approach, components of existing methods, called *method fragments* or *method chucks*, are extracted and stored in a repository known as *Method Base*. Method fragments are subdivided into two main categories: *conceptual* method fragments and *technical* method fragments [6]. A conceptual method fragments is either a *product* fragment, or a *process* fragment. A product fragment describes a product of the information systems engineering process. A process fragment describes the activities carried out to develop a product fragment. *Technical* method fragments describe the automated tools (CASE tools) supporting the information systems engineering process, and are subdivided into three categories: *tool fragment*, *repository fragment* and *process manager fragment*. A tool fragment describes part of the CASE tool functionality. A repository fragment describes part of a data base. A process manager fragment guides the CASE tool user through part of the method, thereby connecting process fragments, product fragments and tool fragments.

The approach to assembly-based Method Engineering presented in [2] aims at constructing a method in order to match as well as possible the situation of the project at hand. It consists in the selection of method fragments from existing methods that satisfy some situational requirements and their assembly. This approach is requirements-driven, meaning that the method engineer must start by eliciting requirements for the method. Next, the method fragments matching these requirements can be retrieved from the Method Base. And finally, the selected fragments are assembled in order to compose a new method or to enhance an existing one. As a consequence, the three key intentions in the assembly-based method engineering process are: *Specify Method Requirements, Select Method Chunks* and *Assemble Method Chunks*. Figure 1 depicts this assembly-based process model for Method Engineering. According to the presented approach, in the last two steps, i.e. selection and assembling the method chunks, semantics of methods need to be considered.

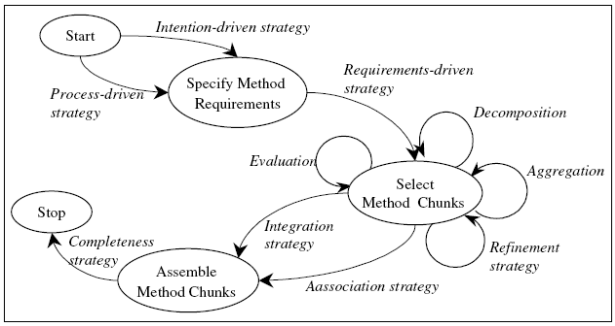

Fig 1. Assembly-based Method Engineering (adopted from [2])

For example, deciding whether two chunks can be assembled to a new method requires the semantic definition of them, because these kinds of decisions cannot be done by means of concrete and syntactical definitions of methods. However selection of suitable method fragments based on method requirements is a major problem in assembly-based method engineering. To overcome this problem, we need to define semantic aspects of methods.

2.2. Tool Support

Method Engineering is such a complex and error-prone process that cannot be properly performed without any automated support. This automated support is provided by Computer-Aided Method Engineering (CAME) environments. A CAME environment is composed of a set of correlated tools aiming to facilitate, in its ideal form, the entire process of Method Engineering. CAME technology dates back to the early days of Method Engineering by the introduction of several academic prototypes. CAME environments are divided into two parts. The CAME part provides facilities for Method Engineering, whereas the CASE part offers facilities for the generation of CASE tools and process managers. The set of Method Engineering tools and the Method Base form the main

elements of the CAME part. The Method Engineering tools offers a set of tools for facilitating the work of method engineers, e.g. for extracting components of existing methods and storing them in the Method Base. The Method Base on which CAME environments are built, forms the kernel of the CAME environment. The obtained method from the CAME part will be given as input to the CASE part. The CASE Generator gets the product part of methods and generates the project specific CASE tool. Process Generator differs from the CASE Generator in that it generates process managers based on the process part of methods. We believe that semantic meta-models are a prerequisite of any CAME environments' Method Base, but a few of the existing CAME environments address this issue. Lack of fragments' semantical aspects leads to complications such as selection and assembling method fragments that may be not semantically composable to a method. However, describing semantics of method fragments is one of the major problems in SME. To overcome this problem, method fragments need to be described in a complete and unambiguous way. Nevertheless, as stated in [6], since methods and their semantics are interpreted differently by different human beings, there is no unique meaning of a method fragment. However, method fragments can be anchored, i.e. described in terms of unambiguously defined concepts and relationships between those concepts, in a system of which the meaning is defined.

The necessity of semantic-based Method Engineering has been stated in previous researches [7, 6, 8, 3]. The term Ontology-Based Method Engineering has been proposed for the first time in [8]. However, a few efforts have been done to consider the semantic aspects of information systems development methods. The first attempt to define semantics of method fragments is done in a work proposed in [7]. In this paper a semantic data-model has been presented from [4], called ASDM, which is applied in *Methodology Representation Model* and its corresponding *Methodology Representation Tool* (MERET). *Methodology Data Model* (MDM) and *process classification system* proposed in [6] are the most comprehensive systems providing facilities to define semantics of method fragments. Decamerone CAME environment provides facilities to define semantics of method fragments. An ontology for product fragments and a process classification system for any method fragments are defined to achieve semantic aspects of method fragments. The proposed ontology is called the MDM which consists of basic concepts of information systems development products and the associations between them. The process classification system employs the notion of goal which is represented as a tuple *(Action, Measure, Product)*. Goals are taken from a process classification, consisting of a set of basic actions in ISD, a set of measures, and a set of product types required in ISD. Basic actions are those actions in ISD with the same effect; a product type is a class of products in ISD with the same purpose, and measure is a qualifier of a product, to indicate temporal state, level of detail, or level of abstraction.

3. Semantic Definition in Method Engineering

In this section, first we summarize some of the efforts done toward the definition of methods' semantic in Method Engineering research area. Afterwards, we propose our ontology for the Method Base.

The first attempt to define semantics of information systems development methods has been proposed in a work done in order to development of a *methodology representation model* and its corresponding CAME environment called *Methodology Representation Tool* (MERET). The semantic notation developed during the MERET project is called ASDM. The next work done toward definition of semantic aspect of methods has been proposed in [6] as an ontology for describing product fragments and a process classification system for describing all of the method fragments.

3.1. ASDM

ASDM provides a rich data structuring capability of modeling objects and the relationships between them. ASDM developed at the Institute for Information Management at the University of St. Gallen. We concentrate briefly on the fundamental concepts in ASDM. ASDM provides a graphical notation and distinguishes between object types and relationship types. An object type provides the description of the structure of a class of individual objects of a certain reality, i.e. instances. This classification of individual objects is a method of abstraction which ignores differences among elements in order to form a generic class. Object types are represented by named boxes. ASDM further provides directed and binary associations between two object types. Associations are distinguished in inheritance, aggregation or horizontal relationships. Horizontal relationships represent a functional dependency between two object types. Horizontal relationship is directed.

An object type can be classified from different points of view. Therefore, the instances of that object type belong to more than one sub type. An inheritance relationship set is a group of inheritance relationships from one object type to others where the object type is classified by one certain view.

3.2. Methodology Data Model and Process Classification System

Methodology Data Model (MDM) proposed as a semantic data-model [6], provides concepts in order to define semantics of method fragments. An ontology for product fragments and a process classification system for any method fragments are defined to achieve semantic aspects of method fragments. The proposed ontology is called the MDM which consists of basic concepts of information systems development products and the associations between them. The process classification system employs the notion of goal which is represented as a tuple (Action, Measure, Product). Goals are taken from a process classification, consisting of a set of basic actions in ISD, a set of measures, and a set of product types required in ISD. Basic actions are those actions in ISD with the same effect; a product type is a class of products in ISD with the same purpose, and measure is a qualifier of a product, to indicate temporal state, level of detail, or level of abstraction. Description of semantics of method fragments is one of the major problems in Method Engineering. To alleviate semantic problems, method fragments are described in terms that are defined as complete and unambiguous as possible. Assembly should also be based on the semantics, and ideally pragmatics, of each method fragment involved, rather than on abstract or concrete syntax. One way to achieve this is to characterize method fragments with as many properties as possible. The problem with this approach is, however, that there are few relationships defined between properties. Moreover, semantics of property value types are in most cases of a rather coarse granularity, which makes them less suitable to provide method fragment semantics. And third, completeness of a description in terms of individually defined properties is hard to prove. Method fragments should be anchored, i.e. described in terms of unambiguously defined concepts and, possibly, associations of an anchoring system Γ. The anchoring function $\alpha: M \rightarrow \gamma\Gamma$, the interpretation function, maps method fragments in M on a subset of the anchoring system. Because mappings need to be unambiguous, α is a bijection function. In principle, the anchoring system prescribes the set of possible method fragments, and is therefore limitative.

An ontology, in the sense of which it is being used in the knowledge-based systems field, is considered an anchoring system with concepts and relationships between those concepts. An ontology for method fragments is an anchoring system $\Delta = <\theta, \psi, \varphi>$ where is a set of unambiguously defined concepts of IS engineering methods, ψ a set of associations, and φ: $\theta \times \psi \rightarrow \theta$ a function relating elements of θ with elements of through an association that is an element of ψ. An ontology may be an abstraction of other ontologies; in this interpretation, the Method Base itself is also an ontology, but for the ME level instead of the ISEM level. Note that we, in contrast to other authors but in accordance with, reserve the term ontology for a structured system, to be as clear as possible and to prevent overloaded terminology. This is the reason why we have introduced the more general term anchoring system, which also includes non-structured systems.

The Methodology Data Model is defined as a structure $\Delta_{MDM} = <CN_0, A_0, \beta>$ with:

- CN_0, the set of MDM concepts,
- A_0, the set of MDM associations, and
- $\beta: CN_0 \times A_0 \rightarrow CN_0$, a function mapping MDM concepts and MDM associations on MDM concepts.

Whereas the MDM only addresses product fragments and repository fragments, because it defines the semantics in terms of structure, the *process classification system* can be used to define any type of method fragment, because it employs the general notion of goal. Goals consist of three components: basic actions, result types (also called product types), and the states a result type can be in (also called measures). A basic action is a class of actions in IS engineering each having the same effect. A result type is a class of products in IS engineering each having the same purpose. A state is a qualifier of a product type, to indicate temporal state, level of detail, or level of abstraction. See [6] if the complete definition is necessary for understanding MDM and its concepts. The second type of anchoring system proposed in [6] is a process classification system. Processes identified apply to any engineering process, be it, for instance industrial engineering, mechanical engineering, or information systems engineering. In order to capture the specific semantics of IS engineering; these processes should be considered in their context, i.e. the results they achieve. This requires, besides a process classification, also a classification of results, often the products delivered by the processes. Anchoring systems formally capture the semantics of method fragments as much as possible. They prescribe the possible relationships between the elementary building blocks of method fragments, and they provide a uniform definition of these building blocks. It is important to notice that there always remains a non-formalisable part in the anchoring system: the definition of the concepts in natural language. It is also important to point out that there are several ways to capture method fragment semantics.

3.3. An Ontology for the Method Base

In this subsection we present our ontology designed specifically for the Method Base. As shown in Figure 2, the root object of this ontology is the *Method Base* itself which is partitioned into *Method fragments* and

Administration. The latter consists of the activities performed to monitor the entire Method Base. Method fragments can be of two main kinds, *Conceptual fragments* and *Technical fragments*. Conceptual fragments itself partitioned into *Product fragments* and *Process fragments*. Relationships between method fragments are of three kinds. *Aggregation*, *Association*, and *Inheritance* relationships, where the latter one demonstrates the Generalization/Specialization relationship between fragments. Technical fragments can be *Tool fragment*, *Process Manager fragment*, or *Repository Fragment.*

Method fragments need to be described in a formal way to enable the automated support of Method Engineering provide by CAME environments. In this regard, method representation languages have been defined. These method representation languages are of three kinds. Most of them are graphical languages which are known as meta-modeling languages. The other languages offer textual constructs in order to represent methods and parts thereof. Some existing languages provide facilities to describe information systems development methods in both graphical and textual manners. Therefore, we defined the Representation Mechanism class in order to state the manner in which methods are described. This class is further partitioned into three subclasses: *Graphical*, *Textual*, and *Graphical/Textual*.

In the proposed approach in [9] method fragments are formalized by a couple <*situation, intention*>, which characterizes the situation that is the input to the fragment process and the intention (the goal) that the fragment achieves.

In this regard, we defined the subclasses *Intention* and *situation* and their corresponding object properties *hasIntention* and *hasSituation* respectively. On the other hand, we have defined another set of classes to describe the activities related to administration of the Method Base. The *Administration* class is further partitioned *Actions* and *Rules*, where the latter describes the rules and constraints defined for the former one. Action class consists of usual database operations such as Store, Retrieve, Update, and Delete. Rules are consisted of any kind of constraint can be defined for method fragments and their relationships. The major rules identified are Connectivity, e.g. *Connectivity* of diagrams, *Consistency* of selected method fragments to constitute in a method, *Conformity* with the meta-model defined for product and process models of methods, *Completeness* of methods and parts thereof.

3.4. Advantages of Ontologies in Method Engineering

Description of semantics of method fragments is one of the major problems in Method Engineering. To alleviate semantic problems, method fragments are described in terms that are defined as complete and unambiguous as possible. Assembly should also be based on the semantics, of each method fragment involved, rather than on abstract or concrete syntax. One way to achieve this is to characterize method fragments with as many properties as possible. The problem with this approach is, however, that there are few relationships defined between properties. Moreover, semantics of property value types are in most cases of a rather coarse granularity, which makes them less suitable to provide method fragment semantics. And third, completeness of a description in terms of individually defined properties is hard to prove [6].

Defining semantics of information systems development methods and their fragments has several advantages. First of all, as mentioned earlier in this paper,

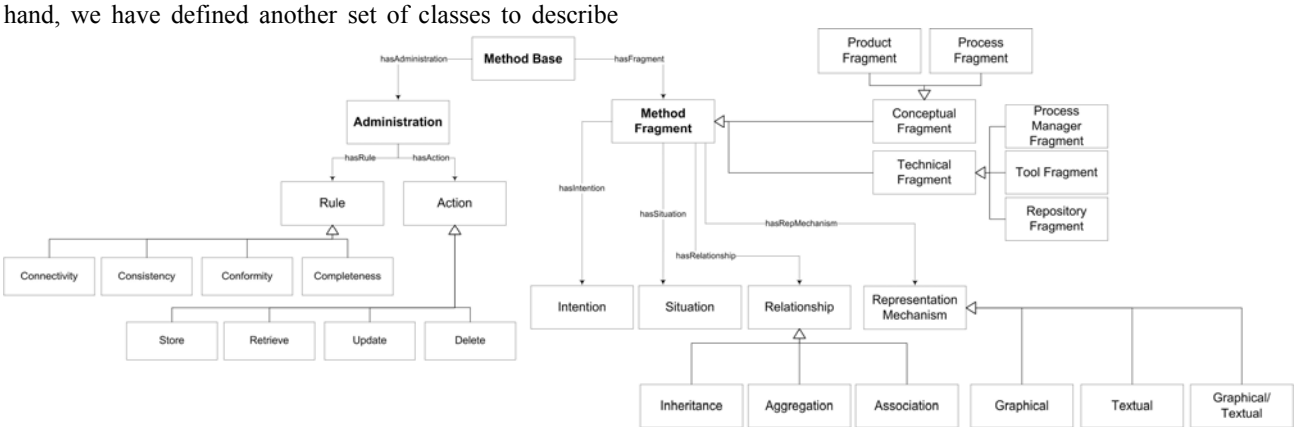

Fig 2. Overview of the proposed ontology

assembly-based Method Engineering is the most popular approach to build new methods. In this regard, selecting the best method fragments fitted to the project-specific method is the hardest part, because selection of method fragments should not be only based on syntactical aspects of method fragments. As a result, by ignoring the semantic aspects of fragments, severe problems occur in selection and assembly of method fragments, e.g. to decide whether

or not two selected method fragments can be assembled to a unique method, we need to access to the semantics of those fragments. Other advantages of the definition of method fragments consist in semantic search of method fragments, checking semantic consistency of fragments, and other checks such as semantic completeness, semantic conformity with the meta-models of product and process models.

4. Conclusion and Future Works

There is some discussion about how ontologies and Software Engineering fit together, and how both communities can learn from each other. We presented the application of ontologies in Method Engineering discipline. As stated in this paper, description of semantics of method fragments is one of the major problems in Method Engineering. Furthermore, in subsection 3.3, we proposed an ontology-based approach to Method Engineering. In this regard, we present a graphical notation of semantic data-model called ASDM, and an ontology for method fragments called MDM. These two semantic systems fall short in addressing the administration issues of the Method Base. However they deal only with method fragments, and do not specifically provide support for the Method Base. Therefore, they cannot be considered as comprehensive ontologies addressing semantics of methods and parts thereof. Our proposed ontology not only addresses the product and process part of methods, but also administration issues of the Method Base. Another advantage of our approach is to conclude another type of method fragments called Technical fragments. In this regard, we developed another class to conclude this concept and three subclasses of it. Our future work includes development of a CAME tool supporting the Hybrid method design approach [5]. We have planned to develop an ontology-based Method Base. In this regard, we have to apply such an ontology proposed in this paper. However, some shortcomings can be considered in our work. For example we did not consider concepts such as actor and role to present the human aspect of method development. A comprehensive ontology for the Method Base needs to address all aspects of the information systems development methods.

Acknowledgments

The authors wish to thank the anonymous reviewers whose feedback helped improve this paper.

References

[1] Gasevic, D., Djuric, D., Devedzi, V. *Model Driven Architecture and Ontology Development.* Springer-Verlag, 2006.

[2] Ralyté, J., Deneckère, R., Rolland, C. *Towards a Generic Model for Situational Method Engineering.* 2003.

[3] Saeki, M. *CAME: The First Step to Automated Method Engineering.* 2003.

[4] Rolland, C. *A Primer for Method Engineering.* Toulouse, France, 1997.

[5] Ramsin, Raman. *The Engineering of an Object-Oriented Software Development Methodology.* Ph.D Dissertation, University of York, 2006.

[6] Harmsen, Anton Frank. *Situational Method Engineering.*Moret Ernst & Young , 1997.

[7] Heym, M., Osterle, H. *A Semantic Data Model for Methodology Engineering.* Amsterdam, North-Holland, 1992. pp. 215-239.

[8] Rosemanna, M., Green, P. *Developing a meta model for the Bunge–Wand–Weber ontological constructs.* 2002. pp. 75-91.

[9] Mirbel, I. *Method Chunk Federation.* EMMSAD'06. 2006.

[10] Happel, H. J., Seedorf, S. *Applications of Ontologies in Software Engineering.* 2004.

[11] Leppanen, Mauri. *An Ontological Framework and a Methodical Skeleton for Method Engineering.* Ph.D dissertation, University of Jyväskylä, 2005.

[12] Leppanen, M. *Conceptual Analysis of Current ME Artifacts in Terms of Coverage: A Contextual Approach.* Paris, France. 2005.

Ali Niknafs M.Sc. student of Software Engineering at Sharif University of Technology, Iran. Currently working on a research project to develop a Computer-Aided Method Engineering environment supporting the Hybrid Method Design approach.

Mohsen Asadi M.Sc. student of Software Engineering at Sharif University of Technology, Iran. He is now working on a research project aiming at extracting Method Engineering process patterns in order to define the generic process model of Method Engineering discipline.

Hassan Abolhassani received his Ph.D. from Saitama University of Japan with a thesis on Automatic Software Design focusing on Learning from Human Designers. His areas of academic research include Software Automation, Semantic web researches, knowledge-based software design, and design patterns.