Representation of Method Fragments
A Comparative Study

Anat Aharoni and Iris Reinhartz-Berger
Department of Management Information Systems,
University of Haifa, Haifa 31905, Israel
anatah@mis.haifa.ac.il, iris@mis.haifa.ac.il

Abstract. The discipline of situational method engineering promotes the idea of retrieving and adapting fragments, rather than complete methodologies, to specific situations. In order to succeed in creating good methodologies that best suit given situations, fragment representation and cataloguing are very important activities. This paper presents and compares three existing approaches to fragment representation. It further provides a set of evaluation criteria for comparing fragment representation approaches. These criteria include expressiveness, consistency, formalism, situational cataloguing, adaptability and flexibility to changes, comprehensibility, and connectivity. Based on this comparison, we introduce a new visual approach that combines the benefits of the three reviewed approaches and attempts to overcome their limitations. This approach relies on a specific domain engineering method, called Application-based DOmain Modeling (ADOM), which enables specification of fragments at various levels of details, specification of fragment types and their constraints, and validation of specific fragments against their relevant fragment types. All these activities are done using a well known modeling language (UML), increasing user accessibility (and consequently comprehensibility).

1 Introduction

As the complexity and variety of computer-based systems have increased, the need for well-defined guidelines that will make the development process most efficient and effective has become crucial. Although sticking to an individual methodology has potential advantages, such as reducing learning and training times and improving the expertise of developers in the chosen methodology, there is no single methodology that can be uniquely pointed as “the best". Furthermore, the possible existence of a universally applicable methodology has been doubted by many researchers, such as [0], and, hence, different types of "local" adaptations and modifications have to be made in order to adjust a methodology to the specific requirements and constraints of a project. The area of *method engineering* [00] aims at providing effective solutions for building, improving, and supporting evolution of development methodologies. *Situational method engineering* [0], which can be viewed as a sub-field of method

Please use the following format when citing this chapter:

Aharoni, A., Reinhartz-Berger, I., 2007, in IFIP International Federation for Information Processing, Volume 244, Situational Method Engineering: Fundamentals and Experiences, eds. Ralyté, J., Brinkkemper, S., Henderson-Sellers B., (Boston Springer), pp. 130-145.

engineering, focuses on creating methodologies especially for specific situations. Both regular and situational method engineering refer to *fragments*, the building blocks of methodologies, rather than to complete methodologies. They offer ways to represent fragments, catalogue them according to different features, retrieve the most appropriate ones to given situations, and organize them into complete methodologies. In order to succeed in creating good situational methodologies, i.e., methodologies that best fit given situations, fragment representation and cataloguing are very important activities. In particular, the fragments have to be represented in a uniform way that includes all the necessary information that may influence their retrieval and assembling. This paper focuses on these activities, presenting and comparing three existing approaches to fragment representation. Based on this comparison, which involves criteria such as expressiveness, consistency, and situational cataloguing, we introduce a new visual approach that combines the benefits of the three reviewed approaches and attempts to overcome their limitations. The contribution of this paper is two folded. First, it provides evaluation criteria for comparing and analyzing situational method engineering approaches that concentrate on fragment representation and cataloguing. To the best of our knowledge, such criteria have not been suggested yet and the comparison of situational method engineering approaches is done based on general method engineering criteria. We use the evaluation criteria for comparing the three reviewed fragment representation methods and for explaining the benefits and limitations of the new introduced approach. Second, the new introduced approach brings further advantages that are not exhibited by the three reviewed representation approaches (or by others): it improves the situational cataloguing ability; it enables constraining and specifying fragment types; and it enables validating the completeness and correctness of fragments (against their fragment types).

The structure of the rest of the paper is as follows. Section 2 motivates the need for situational method engineering. We later use this example for exemplifying the different approaches, their limitations, and advantages. Section 3 lists seven evaluation criteria for comparing fragment representation approaches, while Section 4 uses these criteria for presenting and comparing three particular fragment representation approaches. Section 5 introduces and exemplifies our approach, discussing its benefits and limitations in the light of the other three approaches and the evaluation criteria. Finally, Section 6 concludes and refers to future research plans.

2 The need for situational method engineering: a motivation example

As already noted, situational method engineering deals with creation of methodologies that best fit given situations. The situation can be given as a vector of different properties related to the project, the customer, the developing team, the developing organization, etc. Examples to researches that list such properties can be found at [0, 0]. To motivate the need for situational methodologies, consider the following simple Obsert Oglesby case [0].

Obsert Oglesby is an art dealer who requests an information system to assist him in buying and selling paintings for his gallery. After consulting with an independent consultant, Obsert decided to turn to a well-known development company in order to buy a system which will enable him calculating the minimal and maximal prices of a painting and will also serve in detecting new trends in the art market as soon as possible. The development company which was chosen is familiar with the art world and has developed similar systems. The company mainly works with eXtreme Programming (XP) [0] for small projects which need to be developed quickly in an environment of rapidly changing requirements and with RUP [0] for complex projects which are developed by large teams and require detailed documentation. Since Obsert's case does not completely fit to any of these options, the development team decided to use suitable fragments from both methodologies and to adapt them for the particular case. In order to succeed in this mission, the development team has to tackle three main questions: (1) How to divide a methodology into different fragments that can be reused in various contexts? (2) What are the properties that best characterize each fragment? (3) How (or to what extent) can different fragments be adapted and organized into a complete, consecutive methodology? In the context of fragment representation, these questions can be transformed into the following ones: (1) What are the expressiveness and consistency requirements needed for specifying all kinds of method fragments? (2) What are the situational cataloguing abilities required to be supported at the fragment representation level? (3) How (or to what extent) can the possible adaptation (that a fragment may undergo in a situational methodology) be constrained?

Returning to our Obsert's case, the required system is small, the client (Obsert Oglesby) requests his involvement during the development process, and detailed documentation, especially of the business model and system requirements, is required. Hence, the fragments which may be found as relevant to the early development stages of the requested system are "extract requirements" and "build a business model" from RUP and "on-site customer" from XP. The "extract requirements" fragment may be selected due to the generality of the requirements and their extraction by an external consultant. The “building a business model” may be chosen due to the explicit request of the client to receive a detailed documentation of his business. Finally, the “on-site customer” fragment may be selected due to the client's request to be involved throughout the entire development process. However, since the company has already previous obligations and since the client is relatively small, the "on-site customer” fragment cannot be followed literally. Instead, the company may suggest that the client representative will have the authority and ability to provide information pertaining to the system and to make timely decisions regarding the requirements and their prioritization. However, he/she will not be able to physically present in the development site. This limitation will be overcome by creating a time schedule that defines slots and places for collaboration during the development period.

In order that all these selections and modifications will finalize in a complete, consecutive methodology, the way the fragments are presented and constrained is very crucial. In the next section, we list and elaborate on evaluation criteria for examining and comparing fragment representation approaches.

3 Evaluation criteria for fragment representation approaches

The set of evaluation criteria listed here aims at supporting correct, complete, and consistent representation and cataloguing of method fragments, as well as supporting the successive activities of retrieval, adaptation, and building situational methodologies. These criteria were derived from works on qualities of representation models or languages, especially from [0] and [0].

Expressiveness. Although using the same term, fragments differ from each other. Brinkkemper et al. [0] refer to three orthogonal dimensions when modeling and classifying fragments: perspective, abstraction, and granularity. According to the *perspective* dimension, a fragment can be either product- or process-oriented: product fragments relate to the structural and static aspects of methodologies (e.g., deliverables, documents, and models), whereas process fragments capture the behavioral and procedural aspects of methodologies (e.g., stages, tasks, and activities to be carried out). The *abstraction* level of a fragment can be conceptual or technical: conceptual fragments are descriptions and specifications of methodology parts, while technical fragments are implementations of operational parts of the methodology in the form of tools. Finally, a fragment can reside in one of five possible *granularity* levels: method, stage, model, diagram, or concept. The expressiveness of a fragment representation approach can be measured as how much of this variety of fragment types can be specified using the approach. For specifying process fragments, for example, means for expressing branching, loops, and concurrency are required. For expressing fragments at different granularity levels, encapsulation and generalization mechanisms are required for combining several concepts to one (aggregated or generalized) concept. Furthermore, the relations between different fragments, mainly the interactions between process and product fragments, should be specified somehow.

Consistency. Consistency refers to the fact that the same fragment can be (re)used in different contexts, e.g., while describing a specialized or an aggregated fragment, while defining the relations between process and product fragments, while adapting the fragment to the situation at hand, etc. It is important that all these occurrences of the fragment will be consistent with each other, meaning that changes in one place will be applied to all the other places as well. However, if those changes regard to a specific situation, a separate version of the fragment should be maintained.

Formalism. There are different ways to represent things: graphically, textually, logically, mathematically, etc. Generally speaking, representation formalism is a set of syntactic and semantic conventions that allows describing and specifying things. It can be formal, semi-formal, or completely informal, affecting comprehensibility and non-ambiguity of the specifications. In the context of situational method engineering, the presented fragments have to be retrieved, adapted, and tailored latter and, hence, it is important that their representation will be formal or at least semi-formal.

Situational cataloguing. In order to make fragment retrieval easy, effective, and optionally (semi-)automated, fragment representation approaches should wisely catalogue and index the different fragments according to characteristics and features that may define and distinguish different situations. This criterion checks the ability to describe for each fragment the different organizational, human, and project-related features that best characterize it and are likely to be used for retrieval purposes [0, 0]. These lists of characteristics may be modified over time and location (the developing organization) and may vary when different types of fragments are considered.

Adaptability and flexibility to changes. Situational method engineering mainly deals with two ways for integrating fragments to new methodologies, customization and assembling. *Customization* includes operations that have to be carried out on the original method fragments in order to create new (usually slightly different) versions of that fragments that suit the given situations. *Assembling* deals with attaching and connecting methodology fragments, while transformation and gluing parts between the fragments can be added in order to create complete, consecutive methodologies. This criterion checks the ability to support these operations in the representation level.

Comprehensibility. This criterion checks how easy it is to learn and use the fragment representation approach. This is derived from the approach complexity (number of different concepts), ambiguity, and expected stakeholders (users). Although both regular and situational method engineering are perceived as the responsibility of method engineers only, involving other stakeholders, such as software engineers, developers, and even managers, in the associated method engineering processes and decisions may improve their commitment to the chosen constructed methodologies, so they will actually follow them.

Connectivity. Connectivity measures the ability of the method to tailor fragments derived from different source methodologies [0]. Since different methodologies have different assumptions and characteristics, the ability to represent fragments from various source methodologies is not a trivial task. Furthermore, assembling them to consecutive situational methodologies often requires maintaining transformation and gluing fragments. Although this type of fragments can be analyzed and described in terms of product and process fragments, it has also special requirements which need to be considered by the representation approach, such as storing the associated source and target fragments.

4 Fragment representation approaches

There are several works in the area of method engineering and situational method engineering whose focus is fragment and/or methodology representation. We chose to use three particular approaches in this paper which are consistently cited in the literature and refer to (at least) several of our evaluation criteria, discussed in Section 0. Next, we briefly present each approach, exemplify how it represents the "extract requirements" fragment from RUP, and discuss its benefits and limitations according to the seven evaluation criteria. The main outcomes of this comparison are summarized in the appendix.

4.1 An assembly-based situational method engineering approach

The assembly-based situational method engineering approach [0] aims at supporting the development of web-based Content Management Systems (CMS). The four main stages in this approach are identification of the implementation situation, selection of candidate methods, analysis and storage of relevant fragments in the method base, and assembly of the fragments into a complete methodology using route maps for tuning the fragments to the situation at hand [0]. Regarding fragment representation, the approach uses process-data diagrams, which integrate the process model described by UML activity diagrams with the product model described by UML class diagrams. The relations between these two parts are described by dotted arrows that connect activities with the artifacts they create or adjust. Figure 1 for example, describes the "extract requirements" fragment in this approach.

Several adjustments have been made to the standard UML notation in this approach. First, the approach allows specifying unordered activities. The sub-activities "set priority", "estimate risk", and "set status" in Figure 1, for example, are unordered, but they are all sequential to the sub-activity "categories to functional/nonfunctional requirements". Second, the approach uses three different types of symbols for indicating simple vs. compound concepts. A simple concept, denoted by a rectangle, is atomic and, hence, does not contain other (sub-)concepts. An open concept, denoted by a white shadowed rectangle, consists of a collection of (sub-)concepts. Finally, a closed concept, denoted by a black shadowed rectangle, is an unexpanded compound concept, which consists of (sub-)concepts in other fragments. "Value", "Requirement", and "Priority" in Figure 1 are simple concepts, "Domain Glossary" and "Requirement Document" are open concepts, and Business Model is a closed concept.

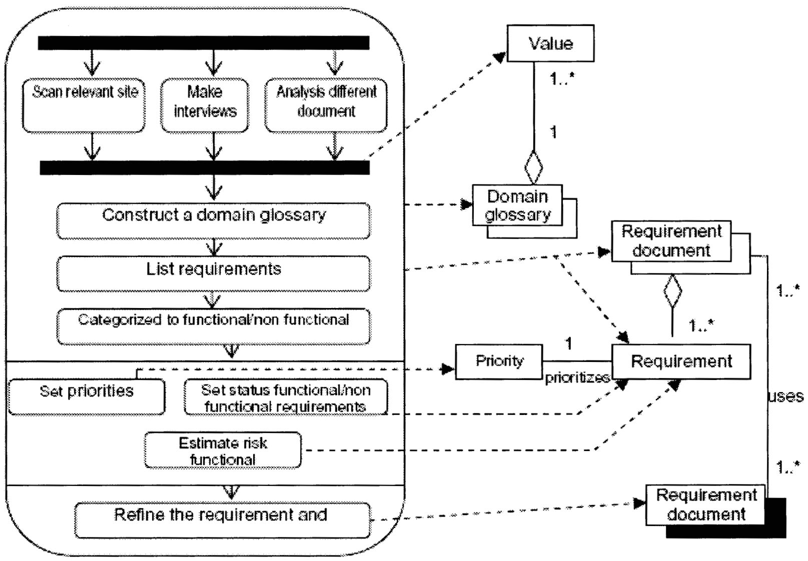


Figure 1. The "extract requirements" fragment of RUP expressed in the assembly-based situational method engineering approach [0].

The approach enables expression of both process and product fragments, as well as the mutual relationships among them. It uses a well known semi-formal, modeling language (UML) with minor changes, which might slightly affect the approach comprehensibility and accessibility. However, it only partially refers to consistency issues by introducing closed concepts, and does not enable specification of the allowed adaptations and changes that a fragment may undergo when assembling or customizing it to a given situation. It further misses the ability to specify the situational features that characterize each fragment, leaving the selection process to the user (i.e., the method engineer).

4.2 The OPEN Process Framework (OPF)

The OPEN Process Framework (OPF) [0] is a large repository for supporting flexible creation of numerous tailored methodologies. Although OPF started as a method engineering approach, we decided to choose it in our research due to its good informational website, its large, diverse, and free repository, and its lately adaptation to situational method engineering requirements.

The OPF consists of three main parts, which are: (1) a repository of reusable method components documented as hierarchical linked Web pages, (2) a meta-model describing the organizational structure of the repository, and (3) construction and usage guidelines. Being a general method engineering approach, OPF does not explicitly refer to method fragments. However, work products, which are significant elements that are described using fields such as contents, stakeholders, and conventions, can be considered as product fragments. Work units, on the other hand, which are described using fields such as completion criteria, tasks, and work products to be produced, can be considered as process fragments. To implement the "extract requirements" fragment, for example, one can consider the following Web pages from the OPF repository: “application requirements engineering” and “business requirements engineering”, which are sub-activities of the “requirements engineering” work unit, and “requirements work product” which is a sub-work product of the “requirements” work product.

The main advantage of this approach is its deeply detailed elements which include a wide variety of aspects that are relevant to each fragment, e.g., goals, preconditions, completion criteria, guidelines, etc. The approach can and does support different types of fragments at different granularity levels. The documentation of the various fragments can also be used for assembling and customizing them into complete methodologies. However, this documentation may be too long, informal (expressed in text), and complex to comprehend and learn to use. Furthermore, the approach does not (semi-)formally support crucial concepts, such as branching, loops, and concurrency, assembling, evolution tracing, or situational cataloguing.

4.3 The scenario-based approach

The scenario-based approach [0] refers to method scenario chunks rather than to fragments. Differently from fragments, a method chunk tightly connects both product and process parts, allowing specifying mutual relations between them. The scenario method base is organized in two levels, the *method knowledge* and the *method meta-knowledge*. The method knowledge level includes for each chunk its interface and body. The *chunk body* describes the product to be delivered and the guidelines to be applied in order to produce the product. The guidelines are defined as hierarchies of related contexts which are connected through three types of links: refinement, composition, and action. The *chunk interface* describes its conditions for applicability to different situations and the ‘intention’ the chunk aims to fulfill. The method meta-knowledge aims at supporting the retrieval process and deals with the context in which method knowledge can be (re)used. This is done by using *chunk descriptors* which express the situation to which the chunk is suitable in terms of the application domains and the design activities in which the chunk can be reused. The chunk interface, body, and descriptor are specified using Standard Generalized Markup Language (SGML) [0].

exemplifies the approach for the "extract requirements" chunk. For the sake of clarity and brevity, we brought here only parts of the SGML description, emphasizing the representation template.

```
(a) <DESCRIPTOR_SITUATION>
   <APPLICATION_DOMAIN> all types of application </APPLICATION_DOMAIN>
   <DESIGN_ACTIVITY> requirements </ DESIGN_ACTIVITY>
    </DESCRIPTOR_SITUATION>
    <DESCRIPTOR_INTENTION>
         <VERB> capture </VERB>
         <TARGET role=<<result>>Type=<<non-scenario-based>>>App. Requirements
         </TARGET>
         <COMPLEX_MANNER>
               <VERB> Produce </VERB>
               <TARGET role=<<result>>Type=<<scenario-based>>...>requirements doc.
               </TARGET>
               <SIMPLE_MANNER> by the requirement workflow of RUP
               </SIMPLE _MANNER>
         </ COMPLEX_MANNER >
     </DESCRIPTOR_ INTENTION >
```

```
(b)
<CHUNK name=<< produce requirements document >>
 type=<<formal>>  informal description= << produce requirements document by obtain an
initial understanding of the domain than draw up an initial set of requirements and finally refine
the requirements artifacts >>  >
    <GRAPHICAL_REPRESENTATION><AHREF=<<fileName.gif>> ></A>
    </GRAPHICAL_REPRESENTATION >
<INTERFACE>
 <CHUNK_SITUATION> <CHUNK_SITUATION>     ...
</CHUNK_INTENTION>          ...          <CHUNK_INTENTION>
</INTERFACE>
 <BODY>
<PRODUCT name=<< Requirements document>>
informal description=<< informal description of the requirements document
structure>>>...
        </ PRODUCT>
<PRODUCT_GRAPHICAL REPRESENTATION>
<A HREF =<<grapich_rep.gif>>>.....
</ GUIDELINE >          ...        <GUIDELINE>
    </BODY>
</CHUNK>
```

Figure 2. The "extract requirements" chunk of RUP expressed in the scenario-based approach: a partial SGML code of (a) the chunk descriptor (b) chunk interface and body

This approach supports specification of method chunks, including their product and process parts, at different granularity levels. It uses a (semi-)formal language in the form of SGML code that might be complex to understand and manage by human users. Each chunk can be reused in a more complex aggregated chunk. Furthermore, the approach enables adapting and changing chunks to specific situations by supporting the definition of parameters within the SGML code. However, the tight coupling between product and process fragments in the approach may cause redundancy and difficulties in reusing the same process or product fragment in different contexts, raising consistency issues that must be handled. Furthermore, at the current stage, the situational cataloguing capabilities of the approach are limited to the application domain and the relevant design activities only.

5 A domain engineering-based approach for fragment representation

As discussed in the previous section, the main limitations of existing method representation approaches are in their user accessibility and comprehensibility, their situational cataloguing abilities, and their ability to constrain the structure and behavior of fragments in order to support a smooth transition to the successive situational method engineering activities (mainly assembling and customization). In order to overcome these limitations, we propose a holistic, visual, domain engineering-based approach for managing, representing, retrieving, customizing, and integrating method fragments in order to create new methodologies that best suit a situation at hand. The fragment representation part of this approach provides the ability to express different types of methodologies and their fragments, their associated characteristics and values, their pre- and post-conditions, and other fragment-related requirements, such as mandatory participants, recommended (optional) participants, triggers, etc. This is done by using a domain engineering approach called Application-based DOmain Modeling (ADOM) and the standard notation of UML 2.0 [0].

Domain engineering [0] is a software engineering discipline concerned with building reusable assets and components that fit to a family of applications, termed a domain. The purpose of domain engineering is to identify, model, construct, catalog, and disseminate a set of software artifacts that can be applied to existing and future software in a particular application domain. As such, it is an important type of software reuse, knowledge representation, and validation. ADOM [0, 0] is a particular domain engineering approach perceiving that applications and domains are similar in many aspects, thus it enables modeling domains with regular software engineering techniques. The application models use domain models mainly for creation (instantiation, reuse) and validation purposes. ADOM is based on a three layered architecture: application, domain, and language. The application layer consists of models of particular applications, including their structure and behavior. The language layer includes meta-models of modeling languages, such as UML. The intermediate domain layer consists of specifications of various domains (i.e., application families). These specifications describe the commonality as well as the variability allowed among applications in the domain. The ADOM approach further enforces constraints among the different layers; in particular, the domain layer enforces constraints on the application layer, while the language layer enforces constraints on both domain and application layers.

ADOM is a quite general architecture and can be applied to different modeling languages that support element classification. ADOM-UML, in which ADOM is used in combination with UML 2.0 [0], was chosen in this context due to the familiarity and establishment of UML in the software development area.

5.1 ADOM-UML

In ADOM-UML, UML stereotypes are used both for classifying application elements according to their relevant domain elements and for specifying the allowed variability among applications in the domain.

In the *language layer*, a new stereotype of the form <<multiplicity min=m max = n>> is defined in order to represent how many times, constrained by the lowest and upper most multiplicity boundaries, a model element of this type can appear in a specific context[1].

In the *domain layer* the main concepts of the domain and the relations among them are specified using UML. The allowed variability within the domain is also specified in this layer by attaching multiplicity stereotypes to the various domain concepts and by adding additional logical constraints (such as "or" to denote variations and "xor" to denote alternatives).

In the *application layer*, the stereotype mechanism is used in order to classify the application elements according to the pre-defined domain elements. The classified application elements are required to fulfill the constraints induced by their classifying domain elements at the domain layer. In addition, the ADOM approach allows adding to application models non-classified elements which are specific to the application at hand and, hence, do not appear in the domain model. These additions are allowed as long as they do not violate the domain constraints.

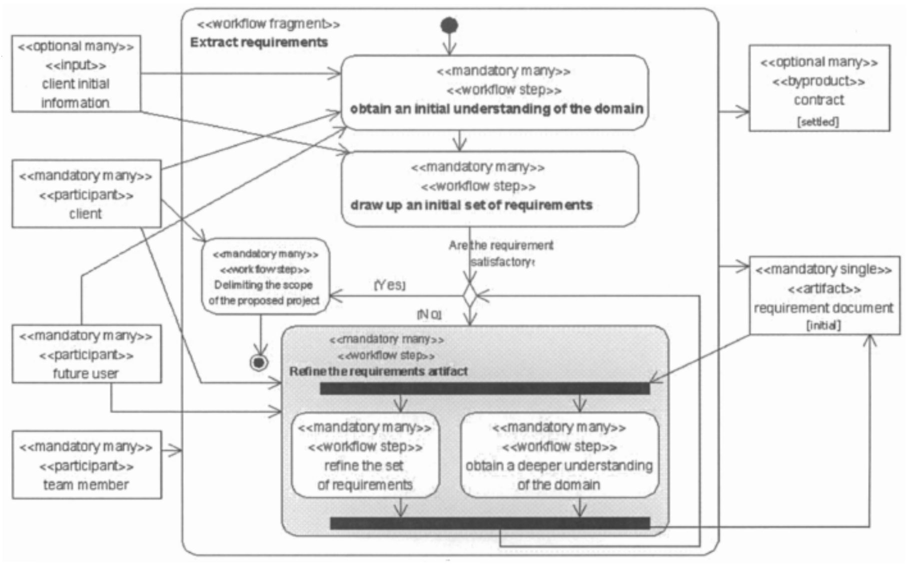
5.2 Representing and cataloguing fragments in ADOM-UML

The structure and guidelines of fragments are described within the domain layer of ADOM, while their instantiations, which specify particular situational methodologies, are defined in the application layer. In these two layers, process and product fragments are respectively described by UML activity and class diagrams, while the lowest (simple, atomic) fragments may link to Web pages, similar to those exist in the OPF repository. The dependencies among process and product fragments can be concluded from the consistency constraints required to be maintained between the relevant class and activity diagrams in UML (e.g., the classes of object nodes that appear in the activity diagrams have to be described in the class diagrams). Furthermore, the different features that characterize each fragment are represented and associated to the fragment models as UML templates, i.e., parameterized elements that can be used to generate other model elements using binding relationships. The exact lists of features that characterize the different types of fragments can be derived from works that were done in the area of situational method engineering, such as [0, 0], and from practitioners.

Figure 3 and Figure 4 respectively exemplify process and product fragments taken from RUP [0]. Figure 3 describes the "extract requirements" process

[1] For clarity purposes, we defined four commonly used multiplicity groups on top of this stereotype: <<optional many>>, where min=0 and max=unbounded, <<optional single>>, where min=0 and max=1, <<mandatory many>>, where min=1 and max=unbounded, and <<mandatory single>>, where min=max=1.

fragment, including its optional inputs, required participants, expected deliverables, skeletal steps and flow of control[2].



```
(b) <SITUATION_CHARACTERISTICS fragmentType=<<process>>
        fragmentName=<<extract requirements>> >
        <PROJECT_CHARACTERISTICS>
                <APPLICATION_DOMAIN>All</APPLICATION_DOMAIN>
                <PROJECT_SIZE>greater or equal 2 sub systems</ PROJECT_SIZE >
                 <FLEXIBILITY_TO_CHANGES>low</ FLEXIBILITY_TO_CHANGES >
                . . .
        </ PROJECT_CHARACTERISTICS>
        <METHOD_CHARACTERISTICS>
                <SOURCE_METHOD>RUP</ SOURCE_METHOD >
                <DEVELOPMENT_ACTIVITY>requirements
                 </DEVELOPMENT_ACTIVITY>
                <PRE-ACTIVITIES>signed contract</ SOURCE_METHOD >
                . . .
        </ METHOD_CHARACTERISTICS>
  </SITUATION_CHARACTERISTICS
```

Figure 3. (a) A description of the "extract requirements" process fragment of RUP in the ADOM-UML-based approach. (b) Its associated characterization file.

Figure 4 describes the "requirement document", which is an artifact that may be produced by the "extract requirements" fragment or another process fragment. The fragment model constrains the general structure of a requirement document, including its possible variability, without referring to its production way. A requirement document, for example, may relate to several business models and business domain glossaries, which are also types of artifacts. Figure 4 also specifies, using UML templates, the situations in which usage of the

[2] Note that UML enables associating separated icons to the various stereotypes in order to help differentiate among them (e.g. humans vs. deliverables). However, in this paper, we preferred using the full (meaningful) stereotype labels so that readers who are not familiar with ADOM will easily understand the models.

"requirement document" product fragment is desirable: the project life cycle is at least one year, the project size is at least two sub systems, and the flexibility to change is low. As this description might become long and embedding it within the graphics may badly affect the comprehensibility of the diagram, we also support the possibility to define the situations to which the fragment is suitable in a separate XML or SGML file. Figure 3 (b) exemplifies such a characterization file for the "extract requirements" process fragment.

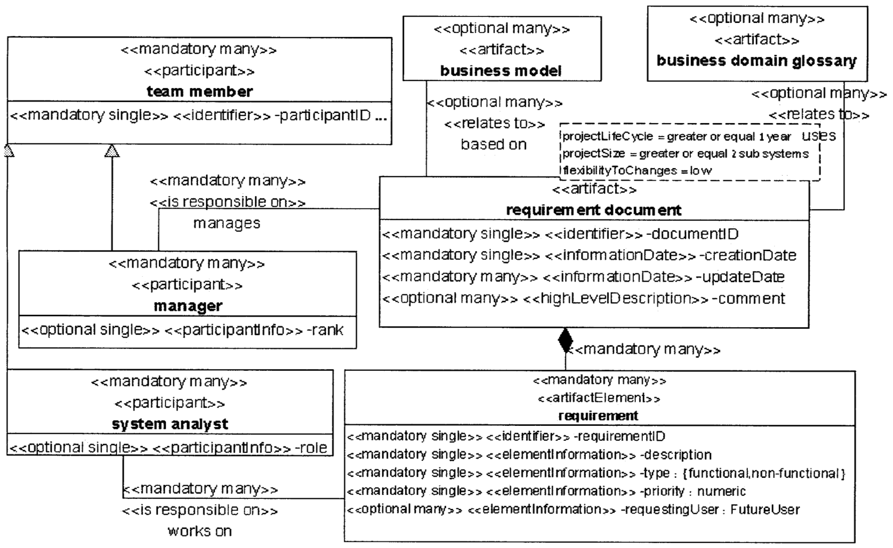

<<mandatory many>>
<<participant>>
team member
<<mandatory single>> <<identifier>> -participantID ...
<<optional many>>
<<artifact>>
business model
<<optional many>>
<<artifact>>
business domain glossary
<<optional many>>
<<relates to>>
based on
<<optional many>>
<<relates to>>
uses
projectLifeCycle = greater or equal 1 year
projectSize = greater or equal 2 sub systems
flexibilityToChanges = low
<<mandatory many>>
<<is responsible on>>
manages
<<artifact>>
requirement document
<<mandatory single>> <<identifier>> -documentID
<<mandatory single>> <<informationDate>> -creationDate
<<mandatory many>> <<informationDate>> -updateDate
<<optional many>> <<highLevelDescription>> -comment
<<mandatory many>>
<<participant>>
manager
<<optional single>> <<participantInfo>> -rank
<<mandatory many>>
<<mandatory many>>
<<participant>>
system analyst
<<optional single>> <<participantInfo>> -role
<<mandatory many>>
<<is responsible on>>
works on
<<mandatory many>>
<<artifactElement>>
requirement
<<mandatory single>> <<identifier>> -requirementID
<<mandatory single>> <<elementInformation>> -description
<<mandatory single>> <<elementInformation>> -type : {functional,non-functional}
<<mandatory single>> <<elementInformation>> -priority : numeric
<<optional many>> <<elementInformation>> -requestingUser : FutureUser

Figure 4. A description of the "requirement document" product fragment of RUP in the ADOM-UML-based approach

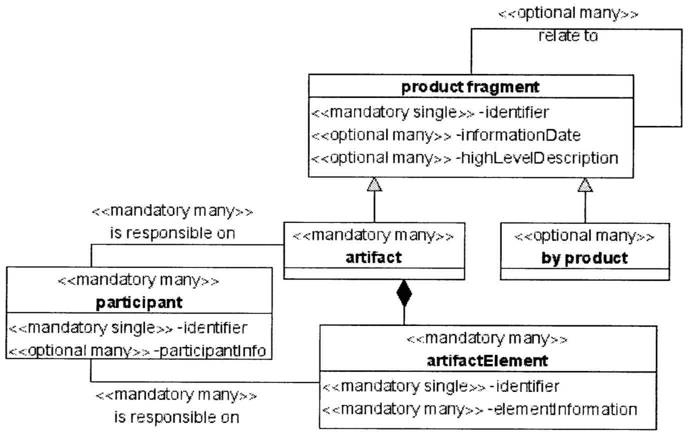

<<optional many>>
relate to
product fragment
<<mandatory single>> -identifier
<<optional many>> -informationDate
<<optional many>> -highLevelDescription
<<mandatory many>>
is responsible on
<<mandatory many>>
artifact
<<optional many>>
by product
<<mandatory many>>
participant
<<mandatory single>> -identifier
<<optional many>> -participantInfo
<<mandatory many>>
artifactElement
<<mandatory single>> -identifier
<<mandatory many>> -elementInformation
<<mandatory many>>
is responsible on

Figure 5. A description of an artifact, which is a specialization of a product fragment, in the ADOM-UML-based approach

Note that all the stereotypes that are used in these diagrams, except from the multiplicity stereotypes discussed earlier, are meaningful concepts in the situational method engineering area. Hence, they can (and may) be generalized and constrained, so that the particular method fragments will be specified in a uniform way. These specifications can be done within ADOM-UML as more general domain models. Figure 5, for example, presents a partial model of an artifact. As can be seen, this meta-model is in yet a more abstract level than the fragment models depicted in Figure 3 and Figure 4, allowing its usage for different kinds of artifacts, e.g., business models and domain glossaries. However, note that the model given in Figure 4 uses the stereotypes defined in Figure 5 and fulfills all the constraints imposed by this figure.

5.3 Analyzing the ADOM-UML-based approach according to the evaluation criteria

Analyzing our fragment representation approach according to the seven aforementioned criteria raises some strengths and limitations that are discussed here and summarized in the appendix, along a comparison with the other fragment representation approaches.

Referring to **expressiveness**, the ADOM-UML-based approach represents both process and product fragments in different granularity levels. The abilities to zoom into activities and to decompose classes in UML are employed in order to specify particular fragments to the required level of details without losing the "big picture" of the fragment as a whole. Furthermore, our approach enables refining the fragment types, such as artifacts and workflow fragments, and representing them in domain models in order to capture the relevant knowledge and to formally constrain the creation of specific fragments of those types. The separation of fragments into different specifications (sometimes expressed by different diagram types) enables using the same fragment in several contexts, e.g., a product that is used by two processes, while preserving autonomy of each part. However, as the fragments might become very complex, this approach also has to deal with visibility problems in the diagrams, both in developing the models and in understanding them. Separating a specification into several diagrams some of which are more specific views of the others is one way to tackle this obstacle.

Regarding **consistency**, the ADOM-UML-based approach allows a fragment to be (re)used in different contexts by different operations and enables managing separated fragment versions according to specific situations. Furthermore, it enables preserving references from derived fragments to their source ones, helping easily identify the reused vs. new fragments, original vs. customized fragments, and the gluing and transformation fragments. In general, the approach provides full support for reuse and composition operators. However, it inherits from UML consistency problems among its diagram types [0].

As for **situational cataloguing**, the ADOM-UML-based approach supports comprehensive and dynamic definition of organizational, human-related, and project-related characteristics, which can be associated to the different fragments and fragment types using UML templates or associated XML files. These features may be used latter for retrieving and assembling the fragments.

Referring to **formalism**, the ADOM-UML-based approach is visual and semi-formal. However, since it applies the well-known modeling language UML, its accessibility to different types of users, such as developers or managers with technical background, is increased over other more formal fragment representation approaches. As noted, the approach accessibility is important for increasing the probability of using the resultant situational methodologies and for making the process of learning and using the fragment representation method easy (earlier referred to as the **comprehensibility** criterion).

Regarding **adaptability and flexibility to changes**, the ADOM-UML-based approach enables all its fragment types to be specialized, adapted, and customized. These operations create new fragments that can be modified as requested by allowing specification of gluing and transformation fragments, customization parts, etc., but without violating the core constraints of the fragment types and of the fragments from which they were derived.

Regarding **connectivity**, the uniform representation of all fragments in the ADOM-UML-based approach enables assembling and connecting fragments that are derived from different source methodologies as long as their pre- and post-conditions fit. Even if they do not exactly fit, the approach allows defining transformation and gluing fragments that help create complete and consecutive situational methodologies.

6 Conclusions and future work

As there is no (and probably will not be) a single universally applicable methodology, the importance of situational method engineering and fragment representation approaches has been increased. In this paper, we listed seven important criteria for evaluating and comparing fragment representation approaches, used them for analyzing the benefits and limitations of three known approaches, and proposed a new approach that aims at overcoming the shortcomings and offering some additional benefits. In the new ADOM-UML-based approach, the fragments are generalized and specified in a domain layer, while the situational methodologies, which assemble and customize the relevant, retrieved fragments, are specified and modeled in the application layer. Due to space limitations, we have not exemplified here a situational methodology, but such an example can be found at [0] along with a description of the supporting CASE tool. Fragment types are also generalized in ADOM as more abstract domain models that guide and constrain the creation of particular fragments of those types. We used UML class and activity diagrams in order to be able to express both product and process fragments and to maintain their consistency. Our comparative analysis shows that the ADOM-UML-based approach supports comprehensive and dynamic definition of characteristics and situational cataloguing information; it better guides the creation of different types of fragments; it is accessible to both method engineers and other potential stakeholders; and it may enable a smooth transition to the successive situational method engineering activities (mainly assembling and customization) by constraining and guiding fragment creation.

As for the future, we plan to elaborate the evaluation criteria to other situational method engineering activities, as well as to show how our extended ADOM-based approach supports these activities in a semi-automatic manner.

References

1. Aydin MN, Harmsen F. Making a Method Work for a Project Situation in the Context of CMM. LNCS 2559, Springer, pp. 158–171, 2002.
2. Brinkkemper, S. Method Engineering: Engineering of information systems development methods and tools. Information and Software Technology, 38(4), pp. 275-280, 1996.

3. Brinkkemper, S. Saeki, M., Harmsen, F. Meta-modelling based assembly techniques for situational method engineering. Information Systems, 24(3), pp. 209-228. 1999.
4. Bryan, M. SGML - An Author's Guide to the Standard Generalized Markup Language. Addison-Wesley publishers Ltd., 1995.
5. Carnegie Mellon Software Engineering Institute. Domain Engineering: A Model-Based Approach, http://www.sei.cmu.edu/domain-engineering , 2002.
6. Extreme Programming Web Site, Extreme Programming: A gentle introduction, http://www.extremeprogramming.org, 2006.
7. IBM, Rational Unified Process, http://www-306.ibm.com/software/awdtools/rup/
8. Krogstie, J. and Arnesen, S. Assessing Enterprise Modeling Languages using a Generic Quality Framework. In J. Krogstie, K. Siau, & T. Halpin, (Eds.), Information Modeling Methods and Methodologies, Idea Group, pp. 63-79, 2005.
9. Krogstie, J., Lindland, O.I., and Sindre, G. Defining Quality Aspects for Conceptual Models. In E. D. Falkenberg, W. Hesse, & A. Olive (Eds.), Proceedings of the IFIP8.1 working conference on Information Systems Concepts (ISCO3): Towards a consolidation of views, pp. 216-231, 1995.
10. Malouin, J.L., Landry, M. The mirage of universal methods in system design. Journal of applied systems analysis, 10, pp. 47-62, 1983.
11. Mirbel, I. Rethinking ISD methods: Fitting project team members profiles. I3S technical report I3S/RR-2004-13-FR, 2004. Available from http://www.i3s.unice.fr/~mirbel/publis/im-isd-04.pdf.
12. Mirbel, I., Method chunk federation. Available at http://www.i3s.unice.fr/~mh/RR/2006/RR-06.04-I.MIRBEL.pdf, 2006.
13. OMG, "Unified Modeling Language: Superstructure", Version 2.0, 2005, http://www.omg.org/docs/formal/05-07-04.pdf
14. OPEN Process Framework (OPF) Web Site. http://www.opfro.org/.
15. Ralyté, J., Deneckere, R., Rolland, C., Towards a generic model for situational method engineering, CAiSE 2003, LNCS 2681, pp. 95-110, 2003.
16. Reinhartz-Berger, I. Conceptual Modeling of Structure and Behavior with UML – The Top Level Object-Oriented Framework (TLOOF) Approach, 24th International Conference on Conceptual Modeling (ER'2005), LNCS 3716, 1-15, 2005.
17. Reinhartz-Berger, I. and Aharoni, A. Representation of Method Fragments: A Domain Engineering Approach. Accepted to the EMMSAD'07 workshop in conjunction with CAiSE'07, 2007.
19. Reinhartz-Berger, I., Sturm, A. Behavioral Domain Analysis – The Application-based Domain Modeling Approach, UML'2004, LNCS 3273, pp. 410-424, 2004.
20. Rolland, C., Plihon, V., Ralyté, J., Specifying the reuse context of scenario method chunks, Proceedings of the CAiSE'98, LNCS 1413, Springer, pp. 191, 1998.
21. Schach, S. R. An Introduction to Object-Oriented Analysis and Design with UML and the Unified Process. McGraw-Hill/Irwin, pp. 56, 2004.
22. Sturm, A., Reinhartz-Berger, I., Applying the Application-based Domain Modeling Approach to UML Structural Views, ER'2004, LNCS 3288, pp. 766-779, 2004.
23. Wistrand, K. Karlsson, F. Method Components – Rationale Revealed. Proceedings of the CAiSE 04, LNCS 3084, Springer, pp. 189-201, 2004.
24. Weerd, I. Brinkkemper, S., Souer, J., Versendaal, J. A situational implementation method for web-based content management system-application: method engineering and validation in practice. Software process: improvement and practice 11(5): 521-538, 2006.

Appendix: The main outcomes from the comparative analysis of the four fragment representation approaches

Criterion	Weerd et al. [0]	OPF [0]	Rolland et al. [0]	ADOM-UML-based approach
Expressiveness	Product and process fragments and the relations among them; different granularity levels; unordered activities and three types of concepts	Product and process fragments; different granularity levels; does not formally support controlling constructs	Method chunks; different granularity levels; supports branching and loops	Product and process fragments; different granularity levels; supports branching, looping and concurrency; preserves references to original fragments
Consistency	Does not support reuse & assembly operations, only the overall route maps contributes to prevent from inconsistencies	Does not support evolution tracing	Each fragment can be reused while aggregating fragments	Full support for reuse and composition operations
Formalism	A visual, semi-formal UML-based language; Some unique adjustments are introduced	A structured hierarchy of Web pages integrated into a visual meta-model	A semi structured, markup language (SGML)	A visual, semi-formal UML- based language; fragment types are specified by domain models
Situational cataloguing	Limited to 7 characteristics that refer to the organization, the technique and the context	No explicit support	Limited to 2 characteristics: application domain and design activity	Supports dynamic lists of characteristics according to the fragment types
Adaptability and flexibility to changes	Supports customization of the process-data diagrams and the route maps are flexible to situational changes	Provides only construction and usage guidelines	Supports parameters	Supports specialization, adaptation, and customization of fragments
Comprehensibility	Only the unique adjustments to UML have to be studied and comprehend	The repository structure helps learn and use the fragments; provides information on stakeholders' involvement	The visual part facilitates fragment usage and learning; does not support stakeholders' involvement	The used language is familiar to the different stakeholders, including method and software engineers
Connectivity	Does not include any rules for connecting fragments; the route maps support assembling of fragments	Does not provide rules or guidelines for connecting fragments	Does not provide rules or guidelines for connecting fragments	Supports customizing and assembling of fragments, as well as specification of transformation and gluing fragments