An Approach for Method Reengineering

Jolita Ralyté *, Colette Rolland **

* CUI, Université de Genève, 24 rue du Gén. Dufour, CH-1211 Genève 4, Switzerland
ralyte@cui.unige.ch
** CRI, Université de Paris 1, 90 rue de Tolbiac, 75013 Paris, France
rolland@univ-paris1.fr

Abstract. The increasing complexity of the Information Systems (IS) asks for new IS development methods constructed 'on the fly' to be adapted to the specific situations of the projects at hand. Situational Method Engineering responds to this need by offering techniques to construct methods by assembling reusable method fragments stored in some method repository. For method engineering to be performed it is necessary to build method bases. In this paper we propose an approach supporting the reengineering of existing methods. The reengineering process leads to the representation of an existing method in a modular fashion i.e. as a set of reusable method chunks, easy to retrieve and to assemble one the others. Once the method chunks are stored in a method repository they can be assembled in different manners to construct new methods. The emphasis of this paper is on the guidance provided by the method reengineering process model. The approach is exemplified with the OOSE reengineering case study.

1. Introduction

More and more real world activities are supported by *Information Systems* (IS). Besides, the complexity of these IS increases whereas the development time reduces. As a consequence, the traditional rigid IS engineering methods are inadequate to provide the necessary support in new IS developments. New methods, more flexible and better adaptable to the situation of every IS development project, must be constructed.

To take this problem into account, *Situational Method Engineering* (SME) proposes to support 'on the fly' construction of methods based on a reuse strategy. By assembling reusable method fragments originating from different methods, a new method can be tailored to the project situation at hand. Works performed in the SME area introduce the notions of *method fragment* [3], [16], [13] and *method chunk* [14], [10] as the basic blocks for constructing 'on the fly' methods. Reusable method fragments/chunks are stored in some method repository [16], [4], [14], [10]. In addition there are a number of proposals for approaches to assemble these fragments/chunks [18], [2], [8], [9], [11]. New methods can thus be constructed by selecting the fragments/chunks that are the most appropriate to a given situation [2], [8] from the method repository. As it can be seen, SME favours the construction of

H.S. Kunii, S. Jajodia, and A. Sølvberg (Eds.): ER 2001, LNCS 2224, pp. 471-484, 2001.

modular methods that can be modified and augmented to meet the requirements of a given situation [3], [17].

Whether attention has been paid to the language for describing method chunks, the question of how to support the method fragments/chunks retrieval and assembly process is not well tackled in the literature. Besides, the prerequisite for modular method construction is a method repository containing a large collection of method chunks. This requires reengineering existing methods to produce the method chunks that populate the method repository. *Method Reengineering* is our concern in this paper.

This paper is organised as follows: section 2 provides an overview of our method reengineering approach. In section 3, we sum up the notion of a method chunk whereas in section 4 we describe the method reengineering process model. Section 5 illustrates the approach with an example demonstrating the reengineering process step by step. Section 6 draws some conclusions and discussions around our future work.

2. Overview of the Approach

The work presented in this paper is a part of our assembly-based method engineering approach summarised in Figure 1. As shown in this figure, the process starts with the reconstruction of the existing IS engineering methods in a modular way. The result is a collection of reusable method chunks, which are stored in the method base. Once the method base is populated with a number of chunks, the construction of a new method is possible through the retrieval of those chunks that match the characteristics of the project situation at hand and their assembly to form the new method.

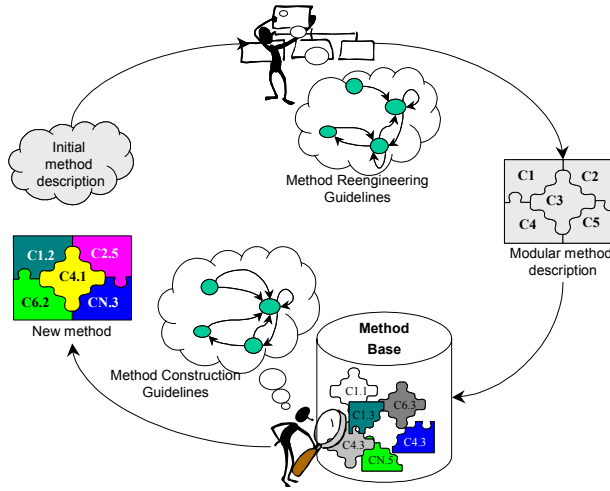

Fig. 1. Assembly based method engineering

In previous papers [14], [10] we presented a *modular method meta-model* allowing to represent any method as an assembly of method chunks. We also proposed a structure for a method chunks repository that we called a method base. Furthermore in

[12], we presented an *assembly process model* to guide the construction of new methods by selecting and assembling method chunks. In this paper we bridge the gap between existing methods and their modular representation. We complete our approach by defining a *method reengineering process model* providing guidelines to reengineer an existing IS development method into reusable method chunks. Figure 2 summarises our method reengineering approach.

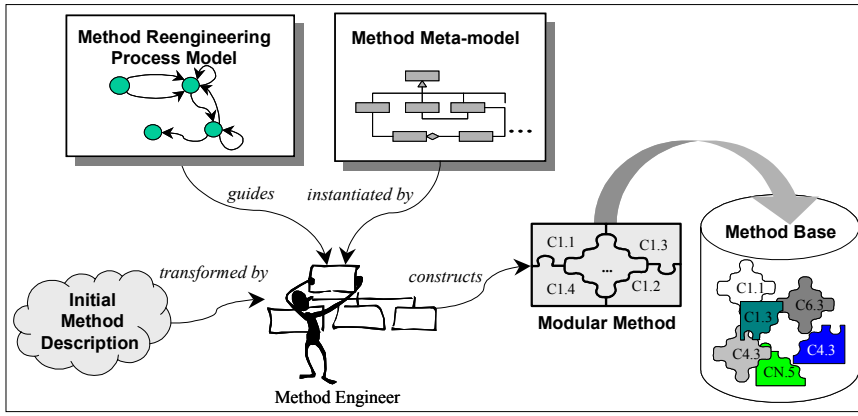

Fig. 2. The approach for method reengineering

As shown in this figure, the central element of our approach is the *method reengineering process model*, which guides the method engineer in the reconstruction of every method into an assembly of method chunks. While guided by this process model, the method engineer instantiates the method meta-model to describe the identified method chunks. In the next section we describe the method meta-model and in section 4 we present the reengineering process in detail.

3. The Notion of a Method Chunk

Situational method engineering proposes to assemble fragments of existing methods to construct a new method. Based on the observation that any method has two interrelated aspects, product and process, several authors propose two types of method fragments: process fragments and product fragments [4], [2]. In our approach we integrate these two aspects in the same fragment that we call a *method chunk*.

A method chunk ensures a tight coupling of some process part and its related product part. It is a coherent module and any method is viewed as a set of loosely coupled method chunks expressed at different levels of granularity [10]. Our modular view of methods favours their adaptation and extension. Moreover, this view permits to reuse chunks of a given method in the construction of new ones. Figure 3 shows the method meta-model (using UML notations [19]). According to this meta-model a method is also viewed as a method chunk of the highest level of granularity. The definition of the method chunk is 'process-driven' in the sense that a chunk is based on the decomposition of the method process model into reusable *guidelines*. Thus, the

core of a method chunk is its guideline to which are attached the associated *product parts* needed to perform the process encapsulated in this guideline.

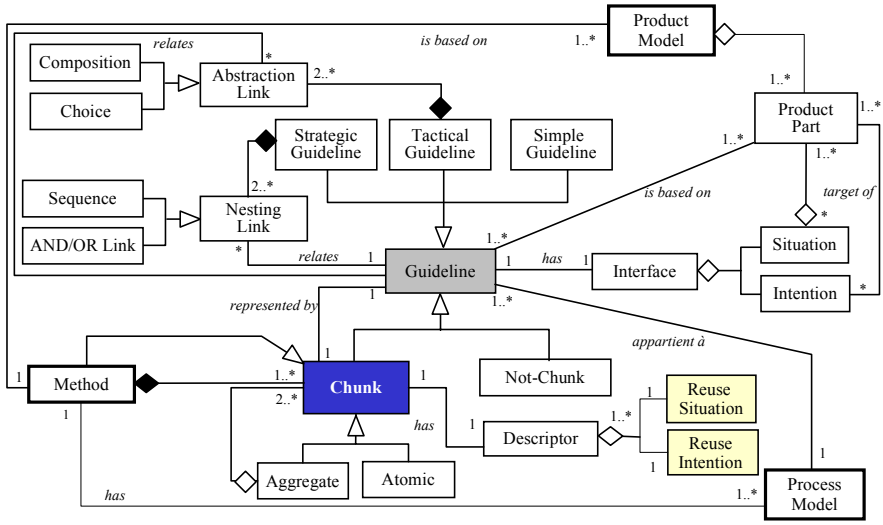

Fig. 3. The method meta-model

A guideline is defined [7] as 'a set of indications on how to proceed to achieve an objective or perform an activity'. For us, a guideline embodies *method knowledge* to guide the application engineer in achieving an intention in a given situation. Therefore, the guideline has an *interface*, which describes the conditions of its applicability (the situation) and a *body* providing guidance to achieve the intention, i.e. to proceed in the construction of the target product. The interface is a couple <*situation, intention*>, which characterises the situation that is the input of the chunk process and the intention (the goal) that the chunk achieves. The body of the guideline details how to apply the chunk to achieve the intention. The interface of the guideline is also the interface of the corresponding method chunk. Guidelines in different methods have different contents, formality, granularity, etc. In order to capture this variety, we identify three types of guidelines: simple, tactical and strategic.

A simple guideline may have an informal content advising on how to proceed to handle the situation in a narrative form. It can be more structured comprising an *executable* plan of actions leading to some transformation of the product under construction.

A *tactical guideline* is a complex guideline, which uses a tree structure to relate its sub-guidelines one with the others. This guideline follows the *NATURE* process modelling formalism [6], which proposes two different structures: the *choice* and the *plan*. Each of its sub-guidelines belongs to one these types of guideline.

A s*trategic guideline* is a complex guideline called a *map* [15], [1], which uses a graph structure to relate its sub-guidelines. Each sub-guideline belongs to one of the three types of guidelines. A s*trategic guideline* provides a strategic view of the development process telling which *intention* can be achieved following which *strategy*. Thus, a map is a labelled directed graph in which the nodes are the intentions and the edges between intentions are strategies. The map permits to represent a

process allowing several different ways to develop the product. A set of guidelines is associated to the map. They help the application engineer to progress in the map and to achieve the intentions following selected strategies. More exactly, a map is a composition of a set of sections where a section is a triplet <*source intention, target intention, strategy*>. An *Intention Achievement Guideline* (IAG) is associated to every section and defines how to realise the target intention from the source intention following the selected strategy. Two other types of guidelines, *Intention Selection Guideline* (ISG) and *Strategy Selection Guideline* (SSG), help to progress in the map i.e. to select the next intention and to select next section respectively.

A *descriptor* is associated to every method chunk. It extends the contextual view captured in the chunk interface to define the context in which the chunk can be reused. The two key elements of the descriptor are the *reuse situation* and the *reuse intention*. Every chunk can be applied in one or several system engineering domains and can support one or more activities in the system design process. The *reuse situation* captures this information in the *Application domain* and *Design activity* attributes. The *reuse intention* expresses the objective that the method chunk helps to satisfy in the corresponding design activity. The descriptor also contains a narrative description of the objective of the chunk and specifies its type (i.e. atomic or aggregate) and identifies the *origin* of the chunk (i.e. the originator method of the chunk). See [10], [11], [14] for more information on the method chunk structure.

4. The Reengineering Process

The process of method reengineering that we propose in this paper makes the assumption that it is worth representing the process model of every method as a map with its associated guidelines. Consequently, method reengineering in our approach consists in redefining the existing method process model in the form of a map and its associated guidelines. The required product parts and descriptors are associated to the each of these guidelines to define them as complete method chunks.

As shown in Figure 4, our method reengineering process model is an instance of the strategic process meta-model (a map) introduced in the previous section and consists in satisfying reengineering intentions using appropriated strategies. We comment the intentions of this map and their associated strategies in turn.

4.1 Map Intentions

As shown in Figure 4, the process model is based on the achievement of four main intentions: *Define a section*, *Define a guideline*, *Identify a method chunk* and *Define a method chunk*.

The first two intentions allow the method engineer to restructure the initial method process model (if existing) or to define it (if the method does not have any formalised process model) as a map. As mentioned in the previous section, a map is made of sections with associated guidelines. Thus, the intention *Define a section* aims at section identification whereas the intention *Define a guideline* refers to the definition of the guidelines associated to these sections.

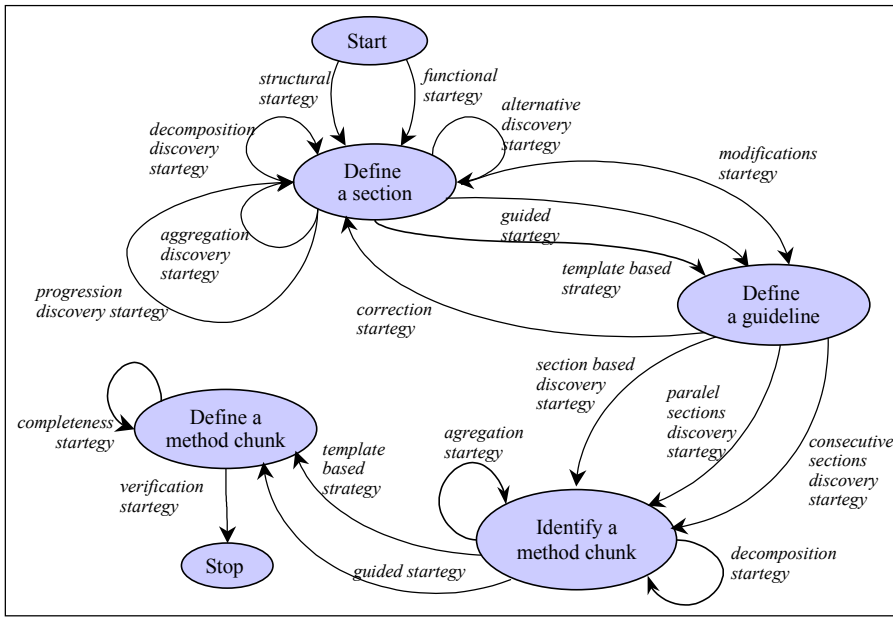

Fig. 4. Method reengineering process model

The two next intentions correspond to the identification and definition of method chunks. This is based on sections and their grouping. Indeed, according to our method chunk concept, every section in the method map is candidate to be defined as a method chunk. The chunk is legitimate if the intention achievement guideline associated to the section identifies an autonomous and reusable methodological procedure. Some aggregations of sections can also form reusable chunks. This reengineering work is supported by the fulfilment of the intention *Identify a method chunk* in Figure 4. The intention *Define a method chunk* supports the definition of the identified method chunks. This includes the completion of the guidelines and the definition of their descriptors.

4.2 Map Strategies

The method reengineering process map presented in Figure 4 proposes a set of strategies to satisfy the four intentions of the map. For example, there are two strategies *structural* and *functional* to achieve the intention *Identify a section*. The *structural strategy* is recommended when the reengineered method does not provide the method engineer with a process model formally defined but rather simply with a description of the product to construct. This strategy uses a glossary of generic process intentions to support the discovery of method intentions. In the contrary, the *functional strategy* should be preferred if the method has a defined process model taking the form of some steps and recommended actions. This strategy helps to identify the method map sections from these actions, and steps.

When the section definition is completed, the method engineer can either define the guidelines associated to these sections (to progress to the intention *Define a guideline*) or to define new sections (to repeat the intention *Define a section*).

The definition of the section guidelines consists in describing the IAG associated to each section, the ISG associated to a set of sections having the same source intention and different target intentions and the SSG associated to every set of parallel sections. The definition of these guidelines is supported by two strategies: the *template based strategy* and the *guided strategy*. The former provides a template for every type of guideline and is advised to experts whereas the latter helps novices by providing more detailed recommendations.

The definition of new sections based on the existing ones may be achieved in four different manners: the *decomposition discovery strategy* helps decomposing an existing section in several ones, the *aggregation discovery strategy* advises how to combine a set of sections into a new one, the *alternative discovery strategy* helps to identify a new section having an alternative strategy or an alternative source or target intention to the existing one, and the *progression discovery strategy* helps to define a new section allowing to progress in the method map from the existing one.

Analysing the already defined guidelines may imply the definition of new sections or the modification of the existing ones. For example, if an intention achievement guideline needs to be decomposed into several sub-guidelines, the corresponding section must also be decomposed. In a similar manner, the modifications (decomposition, aggregation) realised on sections imply modifications on the associated guidelines. The *modification strategy* guides the method engineer to accomplish these transformations.

The identification of method chunks is supported by three strategies: the *section based discovery strategy*, the *parallel sections discovery strategy* and the *consecutive sections discovery strategy*. The first strategy is based on the assumption that every section in the method map may be considered as a method chunk. More exactly, the IAG associated to this section is a basis for a method chunk if it is reusable outside its originator method. According to our method meta-model, the aggregation of the IAG associated to the parallel map sections may be considered as an aggregate chunk too. The *parallel section discovery strategy* helps to identify the IAG associated to parallel sections and to aggregate them into a new guideline. In the same manner, the *consecutive sections discovery strategy* helps to identify the IAG associated to the consecutive map sections and to integrate them with the objective to obtain the guideline of a new aggregate chunk.

Finally, every guideline declared as a reusable one is defined as a method chunk. This is supported by the *template based* and the *guided* strategies that help the method engineer to attach the necessary product parts to the guideline and to define the chunk descriptor. The method reengineering process ends with the *verification strategy.* This strategy helps to verify if all guidelines associated to the map sections have been defined, if all possible combinations of the guidelines have been analysed to identify the method chunks and if all identified chunks have been described. Due to space limitation we cannot present all these guidelines. However some of them will be further explained when used in the case study presented in Section 5. See also [11] for more details.

5. Case Study

In this section we illustrate the method reengineering process model presented in the previous section to reengineer the OOSE method as described in [5]. This method proposes five different models: use case, analysis, design, implementation and test. We restrict our case study to reengineering the use case model construction.

Step 1: Starting the reengineering process. The OOSE description is an informal text describing the structure of the use case model and providing some heuristics " to construct this model". As a consequence, we select the *structural strategy* of Figure 4 to start the reengineering process.

Step 2: Defining the OOSE map sections. The selected strategy recommends first to identify the method map interface, then to identify the intentions and the associated strategies and finally to order them in the map.

The source document makes clear the method goal (the map interface intention) that is *to construct the use case model* of the system under construction starting with the initial *description of the corresponding problem* (the interface situation). The interface of the OOSE method map is therefore as follows:

<(Problem description), Construct the use case model following the OOSE strategy>

To identify the map intentions the guideline suggests to couple the key product parts of the method product model with some of the generic intentions provided in our method base glossary. The use case model includes the following product parts: *actor*, *basic scenario*, *exception scenario* and *use case*. The generic intentions selected from the glossary that seem suitable to those products parts are shown in the table below.

Product part	**Intention verb**
Actor	*Identify, Define*
Basic scenario	*Write, Validate*
Exception scenario	*Write, Validate*
Use case	*Identify, Discover, Conceptualise*

Combining intentions and product parts (verb + target) leads to candidate OOSE map intentions. Based on the OOSE documentation, the final choice of the relevant intentions is made. For example, as the structure of the concept *actor* is very simple (it contains only two attributes: *name* and *informal description*), the intention *Define an actor* can be merged with the intention *Identify an actor*. The OOSE process does not provide any mean to validate the *basic scenario* and the *exception scenario*. Thus, we eliminate the intentions *Validate a basic scenario* and *Validate an exceptional scenario* from the list of candidate OOSE map intentions. The intentions *Identify a use case* and *Discover a use case* are equivalent because the verbs identify and discover are synonyms in the method base glossary. Finally, we select the following list of intentions:

- *Identify an actor*
- *Write a basic scenario*
- *Write an exemption scenario*
- *Discover a use case*
- *Conceptualise a use case*

The next sub-step is to identify the potential strategies to realise these intentions. For this, we need to find the different manners to satisfy these intentions out of the OOSE description. Every identified manner may be considered as a strategy or a tactics. The following table summarises the strategies' identification process.

Intention	Manner (extracted from the OOSE book)	Strategy
Identify an actor	Ask the questions: • Which persons will use one or several functions of the system? • Which persons will handle and maintain the system? • Which external systems will interact with the system ?	***Questions driven strategy***
Write a basic scenario	Write a scenario describing the best understood case of system use.	***Normal case strategy***
Write an exception scenario	Write scenarios describing variations in the basic scenario that correspond to an exceptional system functioning.	***Exception case strategy***
Discover a use case	Ask the questions: • What are the main tasks of each actor? • Does the actor need to read, write or modify the information stored in the system? • Does the actor need to inform the system of external changes?	***Actor based discovery strategy***
Conceptualise a use case	Group the basic scenario with the exception scenarios concerning the same use case.	***Integration strategy***
	Extend the complete use case by other use cases with the objective to represent: • the optional parts of the use case, • the complex and unusual scenarios, • the possibility to introduce new use cases , etc.	***Extension strategy***
	Identify and extract the descriptions, which are common to several use cases, and define them as abstract use cases.	***Abstraction strategy***
	Verify the completeness of the use case model.	***Completeness strategy***

Finally, the guideline suggests to identify the precedence links between the selected intentions. For every intention and one associated strategy we must identify the situation in which the intention may be applied following this strategy. That is, we need to identify the product part necessary to achieve this intention and then to identify the intention constructing this product part. For example, the product necessary to achieve the intention *Identify an actor* following the *question driven strategy* is the *initial description of the problem*. This product part exists when the analysis process starts; therefore the *Identify an actor* intention follows the *Start* intention. The achievement of the intention *Write a basic scenario* is possible only if the corresponding *use case* has been identified. Thus, the intention *Write a basic scenario* follows the intention *Discover a use case*. According to the source description, the *basic scenario* must always be written before the *exception scenarios* are produced. We also know that the use cases discovery is based on actors. Then, the intention *Discover a use case* follows the intention *Identify an actor*. Use case extension and abstraction is a mean to conceptualise new use cases from already conceptualised ones. This means that the *extension* and *abstraction* strategies are reflexive strategies. Figure 5(a) illustrates the result of the initial identification of the OOSE map sections.

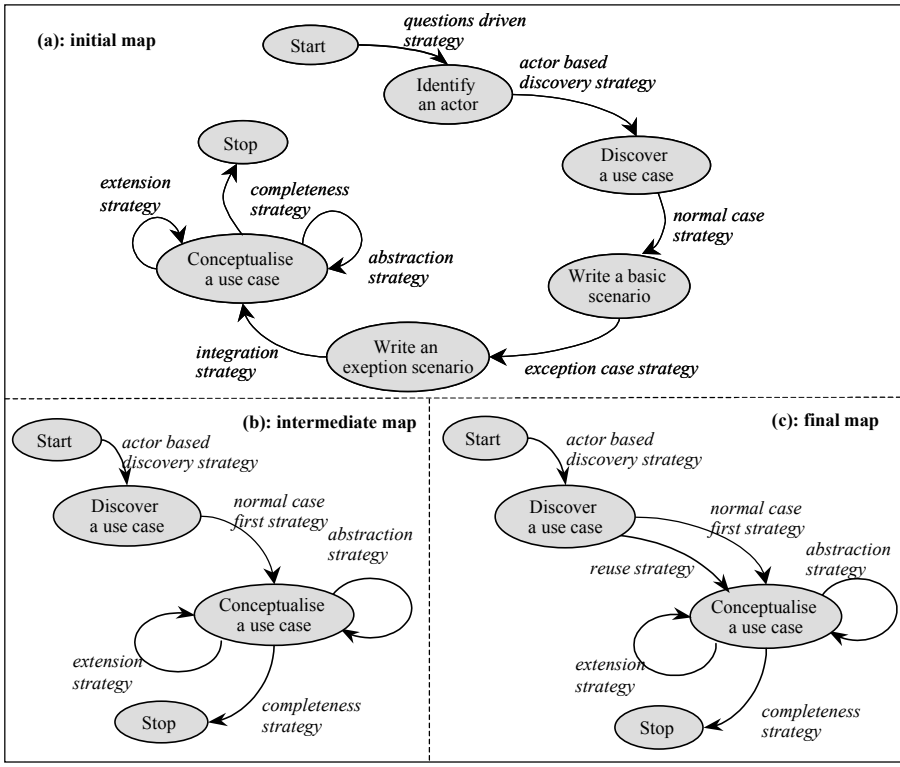

Fig. 5. The construction of the OOSE method map

Step3: Revising the defined sections. The current candidate OOSE map (Figure 5(a)) exhibits an entirely linear use case construction process that suggests some adjustments. Let us use the *aggregation strategy* (Figure 4) to help us combining sections. In the chain of intentions *Start*, *Discover an actor*, *Discover a use case*, only one strategy supports the achievement of every of these intentions. It can be noticed that even defined as a full step in the OOSE method, the identification of actors is not useful as such but only as a mean to identify use cases. For a better reusability of the OOSE approach it is worth transforming the actor identification as a strategy to identify use cases. Thus, we replace the sequence of these sections by the section <*Start, Discover a use case, actor based discovery strategy*> (Figure 5(b)).

In a similar manner we replace the chain of intentions *Write a basic scenario*, *Write an exception scenario* and *Conceptualise a use case* by the section <*Discover a use case, Conceptualise a use case, normal case first strategy*>. This is justified by the fact that only very basic guidelines are provided by the OOSE method for scenario writing, scenario variation discovery and scenario integration in a single use case. This leads us to Figure 5(b). The analysis of this map allows us to notice that *abstract use cases* are generated (following the *abstraction strategy*) but not used. The OOSE method advises to reuse them in the conceptualisation of new use cases but does not say when and how to do it. The *alternative discovery strategy* in the reengineering map (Figure 4) gives us the idea to introduce a *reuse strategy* as a means to achieve

the intention *Conceptualise a use case.* This new strategy guides the reuse of abstract use cases in the description of concrete use cases. Thus, by adding the section *<Discover a use case, Conceptualise a use case, reuse strategy>* we obtain the OOSE map shown in Figure 5(c).

Step 4: Defining guidelines. We can now move on in the reengineering map (Figure 4) to the definition of guidelines. We follow the *guided strategy* to associate an IAG to every section of the OOSE map. Let us consider the IAG associated to the section *<Start, Discover a use case, actor based discovery strategy>*. The definition of an IAG consists in defining its interface and its body. The interface situation refers to the product, which is the target of the source intention, the *Start* intention in our case. The product part is the *"problem description"* and, the interface of the IAG is *<(Problem description), Discover a use case with the actor based discovery strategy>*. The definition of its body depends on its type (simple, tactical or strategic). We identify the IAG under construction as a tactical guideline, which can be represented by a plan including two steps: the definition of actors and the definition of use cases. These two steps are defined as sub-guidelines as shown in Figure 6.

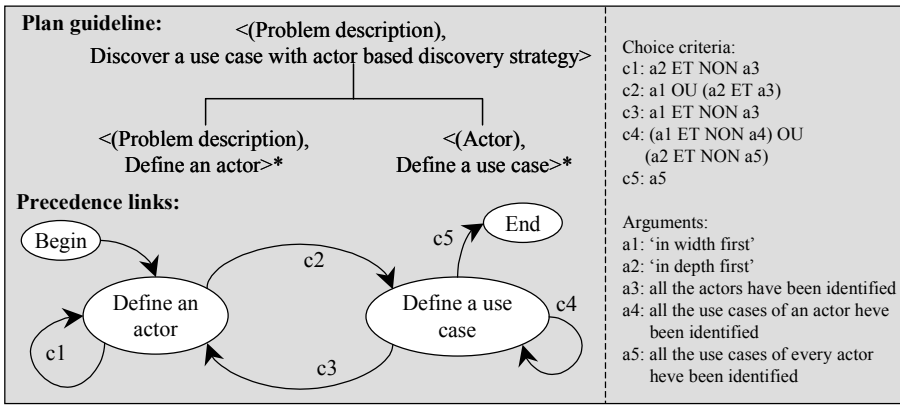

Fig. 6. The IAG associated to the section < Start, Discover a use case, actor based discovery strategy>

Next, we need to define the precedence links between these sub-guidelines. According to the initial method description, the definition of the use cases always follows the identification of the actors. However, one can proceed in different manners, we can identify all the actors first and then define the corresponding use cases or define the corresponding use cases after the identification of every actor. These two ways of proceeding are integrated in the guideline definition through the arguments called *"in width first"* and *"in depth first"* in Figure 6. Moreover, the plan must help verifying if all actors have been identified, if all use cases have been defined for an actor and if all use cases have been defined for all actors. The different combinations of these arguments allow us to define choice criteria of every precedence link of the IAG plan presented in Figure 6. It shall be noticed that the chunk concept contributes to a substantial improvement of the method guideline initially provided by the OOSE description. Following similar approach the two sub-guidelines can be defined.

Step 5: Identifying method chunks. To illustrate the identification of the method chunks in the reengineering process, let us apply the *section based discovery strategy* of the reengineering map (Figure 4) to the OOSE map. This strategy draws the method engineer attention to the fact that a method chunk must satisfy some reusability criteria to be inserted in the method base. This leads to verify if every IAG may be reused as an independent method unit. For example, the IAG associated to the section *<Start, Discover a use case, actor based discovery strategy>* has been transformed enough from its initial version in the OOSE map to be applicable in many different but similar situations requiring to identify the services that an information system must provide to its users. Therefore, we confirm that a method chunk can be based on this IAG.

Let us consider another example to illustrate the *sequence discovery strategy* (Figure 4). Following this strategy we select two consecutive sections *<Start, Discover a use case, actor based discovery strategy>*, *<Discover a use case, Conceptualise a use case, normal case first strategy>* to construct a new guideline embedding the two IAG associated to these two sections of the OOSE map. This guideline is a tactical plan guideline introducing an ordering of the plan elements of the two initial IAG (see Figure 7).

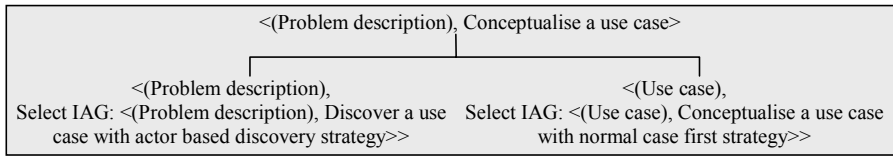

Fig. 7. Example of method chunk

Step 6: Defining method chunks. Finally, we illustrate the definition of a method chunk using the *guided strategy* of the reengineering map (Figure 4). We apply this strategy to the guideline defined in the previous step and presented in Figure 6. The strategy recommends to define first the product parts used by the guideline and then, to define the chunk descriptor. The product parts used in the use case discovery process are the two concepts: *actor* and *use case*. According to the method meta-model presented in Section 3, the definition of the chunk descriptor consists first in determining in which domain and for which design activity the chunk is applicable. This information may be explicitly stated in the method description or inferred from it or from the chunk descriptions. The method base includes a catalogue of predefined domains and design activities that can be selected to characterise a given chunk in the reengineering process. In the example at hand, the chunk may be applied in the following domains: *information systems*, *interactive systems* or *business process reengineering* to support the *discovering of system requirements*. Second, the descriptor intention shall be formulated. It specifies the objective of the chunk and the manner used to attain this objective. In our case, we propose the following: *Discover functional system requirements following a use case discovery strategy*. Third, in order to facilitate the retrieval of the method chunk from the method base it is recommended to complete this information by the informal description of the chunk objective. In our case, the chunk objective is to help the requirements engineer to identify the users of the system and the services that the system must provide. Finally it is recommended to provide the links to the aggregate chunks that include this one as

well as the components of this chunk. The chunk of our example is an atomic one, but it is used in other aggregate chunks that we designate by their interfaces: *<(Problem description), Construct a use case model following the OOSE strategy>* (the whole use case model) and *<(Problem description), Conceptualise a use case>* (the chunk identified in Figure 7). Figure 8 shows the completed method chunk.

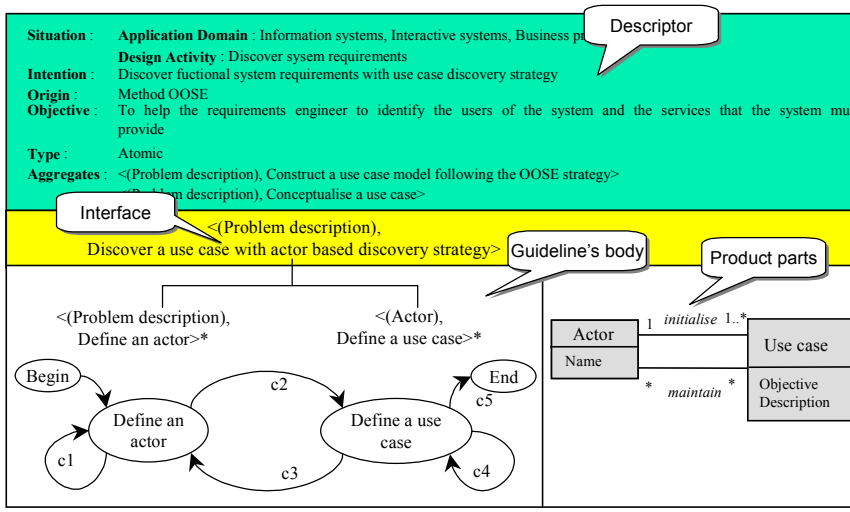

Fig. 8. Example of a completed OOSE method chunk

6. Conclusion

In this paper we look at situational method engineering from the reuse perspective. This method engineering discipline promotes the 'on the fly' method construction by reusing the existing methods' chunks. To enable such method construction we need to build repositories containing different reusable method chunks. Therefore, the existing methods are not presented in a modular way and are not ready to be stored in a method repository. As a solution we propose a process model supporting reengineering of the existing methods in a modular fashion. A method, reconstructed following our reengineering process, is represented by a set of reusable method chunks easy to store in a method repository, to retrieve and to assemble in different manners with the aim to construct new methods.

The reengineering process model is represented as a map with associated guidelines. This allows us to offer flexibility to the method engineer for carrying out the reengineering activity. Besides, guidelines provide a methodological support based on the method meta-model. This meta-model allows to represent any method as a collection of the chunks of different granularity levels where the highest level corresponds to the overall method represented by a strategic guideline i.e. a map with associated guidelines.

Our reengineering process has been evaluated on different methods. The obtained results are encouraging and the experience is positive. In this paper we illustrate the

application of this process on the OOSE method. A software environment to support our reengineering process and to improve its effectiveness is our current preoccupation.

References

1. Benjamen A., *Une Approche Multi-démarches pour la modélisation des démarches méthodologiques*. Thèse de doctorat en informatique de l'Université Paris 1, 1999.
2. Brinkkemper S., M. Saeki, F. Harmsen, *Assembly Techniques for Method Engineering*. 10th Conference on Advanced Information Systems Engineering, CAiSE'98. Pisa Italy, 1998.
3. Harmsen A.F., S. Brinkkemper, H. Oei, *Situational Method Engineering for Information System Projects*. In Olle T.W. and A.A. Verrijn Stuart (Eds.), Mathods and Associated Tools for the Information Systems Life Cycle, Proc. of the IFIP WG8.1 Working Conference CRIS'94, pp. 169-194, North-Holland, Amsterdam, 1994.
4. Harmsen A.F., *Situational Method Engineering*. Moret Ernst & Young , 1997.
5. Jacobson I., M. Christenson, P. Jonsson, G. Oevergaard, *Object Oriented Software Engineering: a Use Case Driven Approach*. Addison-Wesley, 1992.
6. Jarke M., C. Rolland, A. Sutcliffe, R. Domges, *The NATURE requirements Engineering*. Shaker Verlag, Aachen 1999.
7. Le Petit Robert, French Dictionary, Dictionnaires LE ROBERT, France,1995.
8. Plihon V., J. Ralyté, A. Benjamen, N.A.M. Maiden, A. Sutcliffe, E. Dubois, P. Heymans, *A Reuse-Oriented Approach for the Construction of Scenario Based Methods.* 5th International Conference on Software Process (ICSP'98), Chicago, Illinois, USA, 1998.
9. Ralyté J., C. Rolland, V. Plihon, *Method Enhancement by Scenario Based Techniques*. 11th Conference on Advanced Information Systems Engineering CAiSE'99, Germany, 1999.
10. Ralyté J., *Reusing Scenario Based Approaches in Requirement Engineering Methods: CREWS Method Base*. Proc. of the 10th Int. Workshop on Database and Expert Systems Applications (DEXA'99), 1st Int. REP'99 Workshop, Florence, Italy, 1999.
11. Ralyté J., *Ingénierie des méthodes par assemblage de composants*. Thèse de doctorat en informatique de l'Université Paris 1. Janvier, 2001.
12. Ralyté J. C. Rolland, *An Assembly Process Model for Method Engineering*. 13th Conf. on Advanced Information Systems Engineering, CAISE'01 Interlaken, Switzerland, 2001.
13. Rolland C., N. Prakash, *A proposal for context-specific method engineering*, IFIP WG 8.1 Conf. on Method Engineering, pp 191-208, Atlanta, Gerorgie, USA, 1996.
14. Rolland C., V. Plihon, J. Ralyté, *Specifying the reuse context of scenario method chunks*. 10th Conf. on Advanced Information Systems Engineering, CAiSE'98. Pisa Italy, 1998.
15. Rolland C., N. Prakash, A. Benjamen, *A multi-model view of process modelling*. Requirements Engineering Journal, p. 169-187,1999.
16. Saeki M., K. Iguchi, K Wen-yin, M Shinohara, *A meta-model for representing software specification & design methods.* Proc. of the IFIP¨WG8.1 Conference on Information Systems Development Process, Come, pp 149-166, 1993.
17. van Slooten K., S. Brinkkemper, *A Method Engineering Approach to Information Systems Development.* In Information Systems Development process, N. Prakash, C. Rolland, B. Pernici (Eds.), Elsevier Science Publishers B.V. (North-Holand), 1993.
18. Song X., *A Framework for Understanding the Integration of Design Methodologies.* In: ACM SIGSOFT Software Engineering Notes, 20 (1), pp. 46-54, 1995.
19. Rational Software Corporation, *Unified Modelling Language version 1.3*. Available at http://www.rational.com/uml/resources/documentation/, 2000.