An Assembly Process Model for Method Engineering

Jolita Ralyté[1,2] and Colette Rolland[1]

[1] CRI, Université Paris 1 Sorbonne, 90, rue de Tolbiac, 75013 Paris, France
[2] MATIS, Université de Genève, rue du Gén. Dufour 24, 1211 Genève, Switzerland
{ralyte, rolland}@univ-paris1.fr

Abstract. The need for a better productivity of system engineering teams, as well as a better quality of products motivates the development of solutions to adapt methods to the project situation at hand. This is known as situational method engineering. In this paper we propose a generic process model to support the construction of a new method by assembling method chunks generated from different methods that are stored in a method base. The emphasis is on the guidance provided by the process model, as well as on the means underlying guidelines such as similarity measures and assembly operators. The process model is exemplified with a case study.

1 Introduction

We are concerned with *Situational Method Engineering* (SME). SME aims at defining information systems development methods by reusing and assembling different existing method fragments. The term *method fragment* was coined by Harmsen in [5] by analogy with the notion of a software component. Similarly to the component driven construction of software systems, SME promotes the construction of a method by assembling reusable method fragments stored in some method base [20], [6], [16], [13]. As a consequence SME, favours the construction of modular methods that can be modified and augmented to meet the requirements of a given situation [5], [21]. Therefore, a method is viewed as a collection of method fragments that we prefer to call *method chunks* [15], [13] to emphasise the coherency and autonomy of such method modules. New methods can be constructed by selecting fragments/chunks from different methods, which are the most appropriate to a given situation [3], [10]. Thus, method fragments/chunks are the basic building blocks, which allow to construct methods in a modular way.

The objective of our work is to propose a complete approach for method engineering based on a method chunk assembly technique. In previous papers [16], [13] we presented a *modular method meta-model* allowing to represent any method as an assembly of the reusable method chunks. In this paper we are dealing with the method chunk assembly process. We present a generic process model, the *Assembly Process Model* (APM), to guide the assembly of method chunks using different strategies depending on the type of situation in which the assembly activity has to be carried out. Chunk assembly is the support of situational method engineering and therefore we propose a *Method Engineering Process Model* (MEPM) providing several different ways to assemble chunks with the objective of constructing new methods or enhancing the existing methods by new models and/or new ways of working. Whereas the APM views the assembly of method chunks 'in the small', the MCPM takes a broader view where assembling method chunks is part of a larger method engineering process. As a consequence, the APM is embedded in the MEPM.

Both process models, namely the APM and the MEPM, are expressed using the same notations provided by a process meta-model. A *process meta-model* is an abstraction of different process models, i.e. a process model is an instance of a process meta-model. In this paper, we use the strategic process meta-model presented in [19] and [1]. Following this meta-model, a process model is presented as a *map* and a set of associated *guidelines.* Such representation of the process model allows us to provide a strategic view of different processes. Indeed, this view tells what can be achieved - the *intention*, and which *strategy* can be employed to achieve it. We separate the strategic aspect from the tactical aspect by representing the former in the method map and embodying the latter in the guidelines. By associating the guidelines with the map, a smooth integration of the strategic and the tactical aspects is achieved.

This paper is organised as follows: section 2 highlights the need for different strategies for assembling method chunks to form a new method and motivates different ways of method engineering based on method chunk assembly. The former is encapsulated in the APM whereas the latter is captured in the MEPM. In section 3, we take the view of method engineering 'in the large' and present the MEPM. The MEPM includes the APM, which is presented in section 4. Section 5 illustrates the approach with an example demonstrating the process step by step. Section 6 draws some conclusions around our work.

2 Chunk Assemblies and Method Engineering

The attempts to define assembly processes [3], [11], [22] highlight the assembly of method fragments as rather independent and supplementary to one another. A typical example would be to adding a given way of working some new activity borrowed from another method and/or adding to the product model of one method a new concept borrowed from another method. In such a case, the assembly mainly consists in establishing links between the 'old' elements and the 'new', added ones. We found cases quite different where elements to assemble are overlapping. This led us to the identification of two assembly strategies: the *association* and the *integration.*

As shown in Figure 1, the first strategy is relevant when the method chunks to assemble do not have elements in common. This might occur when the end product of one chunk is used as a source product by the second chunk. For example, the chunk producing use cases and the chunk constructing the system object structure can be assembled to get a method with a larger coverage than any of the two initial ones. The assembly process is therefore mainly dealing in making the bridge. The second strategy is relevant to assemble chunks that have similar engineering objectives but provide different ways to satisfying it. In such a case, the process and product models are overlapping and the assembly process consists in merging overlapping elements. The integration strategy will be necessary, for example, to assemble two different chunks dealing both with a use case model construction. These two strategies are embedded in the APM presented in section 4.

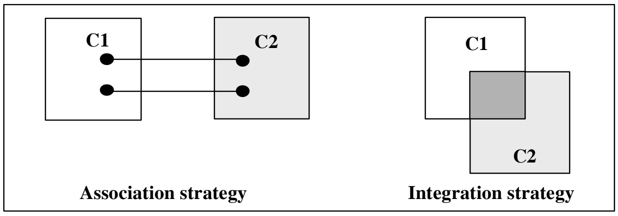

Fig. 1. Strategies to assemble method chunks

The assembly of method chunks is a means, a technique to construct a method in a reuse driven fashion. However, method construction is not restricted to chunk assembly. It includes, for example, the elicitation of requirements for the method to construct, amend or enhance. Besides, the ways the assembly technique will be used differ depending of the objective assigned to the situational method engineering project. There are many different reasons for constructing a new method. We identified three of them:

1. To define a brand new method to satisfy a set of situational requirements;
2. To add alternative ways-of-working in a method to its original one;
3. To extend a method by a new functionality.

Each of these delineates a specific strategy for method engineering that we have embedded in the MEPM. The first strategy is relevant in situations where either there is no method in use or the one in use is irrelevant for the project (or class of projects) at hand. The second strategy is relevant when the method in use is strong from the product point of view but weak from the process viewpoint. Enhancing the process model of the existing method by one or several new ways of working is thus the key motivation for method engineering. The third strategy is required in situations where the project at hand implies to add a new functionality to the existing method which is relevant in its other aspects. We present the MEPM in the next section.

3 The Method Engineering Process Model (MEPM)

Figure 2 shows our proposal to engineer a method through an assembly of method chunks. We use the strategic process meta-model [19], [1] to represent our process model as a *map* with associated *guidelines*. A *map* is a directed labelled graph with *intentions* as nodes and *strategies* as edges. The core notion of a map is the *section* defined as a triplet <*source intention, target intention, strategy*>. A map includes two specific strategies, *Start* and *Stop* to start and stop the process, respectively. As illustrated in Figure 2, there are several paths from *Start* to *Stop*. A map therefore includes several process models that are selected dynamically when the process proceeds, depending on the current situation. Each section is associated to a *guideline* that provides advice to fulfil the target intention following the section strategy. Furthermore, a section can be *refined* as an entire map at a lower level of granularity.

The MEPM represented by a map in Figure 2 includes two intentions *Specify method requirement*s and *Construct method*. The latter corresponds to the method engineering's essential goal, whereas the former is the prerequisite for the latter. The formulation of this intention, *Specify method requirement*s means that our approach is requirements-driven. In order to construct a new method, we propose to start by eliciting the requirements for the method engineering activity.

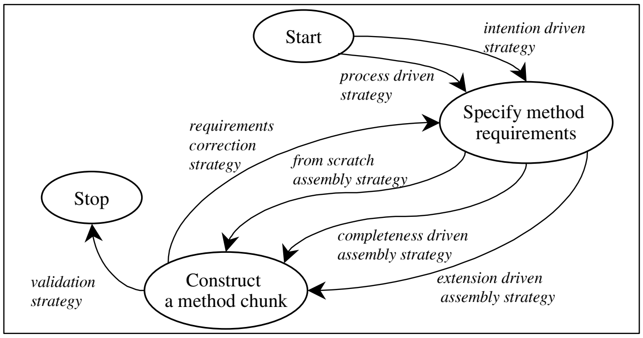
Start
intention driven strategy
process driven strategy
Specify method requirements
requirements correction strategy
from scratch assembly strategy
Stop
completeness driven assembly strategy
validation strategy
Construct a method chunk
extension driven assembly strategy

Fig. 2. The method engineering map (MEPM)

The map identifies two different strategies to *Specify method requirement*s, namely the *intention driven strategy* and the *process driven strategy*. Both lead to a set of requirements expressed as a map that we call the *requirements map*. However, each strategy corresponds to a different way of eliciting the requirements. The former is based on the inventory of engineering goals whereas the latter infers these goals from an analysis of the engineering activities that must be supported by the method.

Once the requirements have been elicited, the intention *Construct method* can be achieved. The MEPM proposes three different assembly strategies: *from scratch*, *enhancement driven* and *extension driven*, to help the method engineer to achieve this intention. The three strategies correspond to the three method engineering situations that were identified and motivated in the previous section. The *from scratch assembly strategy* corresponds to situations where a brand new method has to be developed, whereas the two others, *enhancement driven assembly strategy* and *extension driven assembly strategy*, are relevant when a method already exists. As indicated by the names of the strategies, the three proposed ways are based on a method chunk assembly technique developed in the next section. Backtracking to the requirements definition is possible thanks to *the requirements correction strategy*. Finally, the *validation strategy* helps verifying that the assembly of the selected method chunks satisfies all requirements and ends the method engineering process.

According to the map meta-model used above to model the MEPM, each section of the map is associated to a *guideline* providing advice on how to proceed to achieve the target intention and can be *refined* as a map of a finer level of abstraction. For the sake of space we do not present the guidelines associated to every section of the MEPM (see [14] for details) but concentrate on the refinement of the section <*Specify method requirements, Construct method, from scratch assembly strategy*> dealing with the assembly of method chunks. The refined map of this section models the method chunk assembly process. The APM is presented in detail in the next section.

4 The Process Model for Method Chunk Assembly (APM)

This section presents the *assembly process model* guiding the selection of method chunks matching a set of situational requirements and their assembly to form a new method. It is a generic process model in the sense that it includes a number of strategies to retrieve and assemble chunks providing solutions for the different engineering situations the method engineer may be faced with. In particular the map includes two strategies *(integration strategy and association strategy*) to assemble chunks that we identified from the literature and case studies and introduced in section 2. The APM is presented as a map, the *assembly map*, in Figure 3.

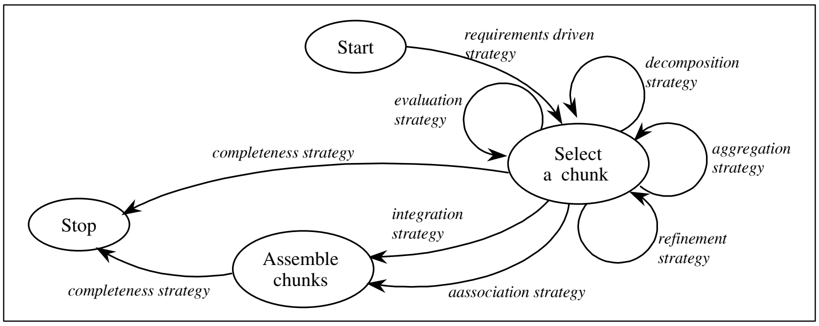

Fig. 3. The assembly map (APM)

As shown in Figure 3, the APM proposes several different ways to select chunks matching requirements as well as different strategies to assemble them. It is based on the achievement of two key intentions: *Select a chunk* and *Assemble chunks*. The achievement of the first intention leads to the selection of chunks that match the requirements from the method base. The second intention is satisfied when the selected chunks have been assembled in a coherent manner.

The process starts by the selection of the *requirements driven strategy*. The associated guideline helps the method engineer to select candidate chunks that are expected to match the requirements expressed in the requirements map (the result of the achievement of the intention *Specify method requirements* in MEPM). It suggests to formulate queries to the method base giving values to the attributes of the descriptors and interfaces of chunks (see [13], [14]) to identify the chunks that are likely to match part or the totality of the requirements map.

Any time a chunk has been retrieved, the assembly map suggests to validate this candidate chunk by applying the *evaluation strategy*. The *evaluation strategy* helps in evaluating the degree of matching of the candidate chunk to the requirements. This is based on similarity measures between the requirements map and the map of the selected chunk. We present these similarity measures in section 4.1.

The *decomposition, aggregation and refinement* strategies help to refine the candidate chunk selection by analysing more in depth if the chunk matches the requirements. The *decomposition* strategy is relevant when the selected method chunk is an aggregate one having some component parts that may not be required. This strategy helps to select those, which are the adequate ones. The *aggregation* strategy is relevant when the candidate chunk partly covers the requirements. This strategy suggests to search for an aggregate chunk containing the candidate chunk based on the assumption that the aggregate method chunk might provide a solution for the missing requirements. The *refinement* strategy proposes to search for another chunk satisfying the same intention but providing a set of guidelines richer than those of the candidate chunk.

When at least two chunks have been selected, the method engineer can progress to the assembly of these chunks following one of two strategies, namely the *integration strategy* and the *association strategy*. As discussed in section 2.1, the choice of the strategy depends of the presence /absence of overlap between the chunks to assemble. If the two chunks help to achieve the same intention in the system engineering process and to construct the same or a similar product, the *integration strategy* must be chosen. If the selected chunks do not overlap in terms of intention to achieve and product to construct, then the *association strategy* must be selected. In the first case where chunks partially overlap, the integration of these chunks produces a new method whose product and process models are 'richer' than those of the initial chunks. With the second strategy for assembling chunks dealing with different aspects of system design, and supplementing one another, the result is a new method providing a larger coverage of design activities. We present the two strategies in more detail in sections 4.2 and 4.3. The two assembly strategies use *assembly operators*, *similarity measures* and *quality validation rules* [12], [14]. Their use will be exemplified in section 5. Just as a reminder, let us mention that there are two types of operators to assemble process models parts and product models parts, respectively. The similarity measures are used to compare chunks before their assembly and to identify whether they are overlapping. This will help to choose the right strategy between the *integration strategy* and the *association strategy*.

In order to check if the current chunk assembly matches the requirements, the method engineer shall use the *completeness strategy*. If the response is a positive one, the assembly process ends. In the reverse case, other chunks have to be selected and assembled to gain the required method completeness.

4.1 Similarity Measures

Measures to estimate the similarity of conceptual schemas have been proposed by several authors, for different purposes: in order to identify reusable components [4] and to select these components [8]. Generally, these approaches measure the closeness between entities of different conceptual schemas by evaluating the common properties and links with other entities [4]. The global complexity of the schemas is also taken into account. Bianco et al. [2] proposes similarity metrics to analyse heterogeneous data base schemas. These metrics are based on the semantic and structural similarity of elements of these schemas. In our approach we use measures inspired from those proposed by Castano [4] and Bianco et al.[2]. We distinguish two types of measures: those, which allow to measure the similarity of the elements of *product models* and those which allow to measure the closeness of *process models* elements. We present them in turn.

Product Models Similarity Measures. We use semantic and structural measures to compare elements of product models.

The *Name Affinity (NA)* metric allows us to measure the semantic similarity of concepts belonging to different product models. To apply this measure, the concepts of the chunks must be defined in a thesaurus of terms. Moreover, the synonymy (SYN) and the hyperonymy (HYPER) relationships between these concepts must be defined. The SYN relation connects the terms t_i and t_j ($t_i \neq t_j$) which are considered as synonyms. This is a symmetrical relation. For example, <*Goal SYN Objective*>. The HYPER relation connects two terms t_i and t_j ($t_i \neq t_j$) where t_i is more general than t_j. This is not a symmetrical relation. The inverse relation is the hyponymy (HYPO). For example, <*Scenario HYPER Exceptional scenario*>.The thesaurus of terms is a network where the nodes are the terms and the edges between the nodes are the terminological relations. Every terminological relation R ∈{SYN , HYPER /HYPO} has a weight σ_R. For example, $\sigma_{SYN} = 1$ and $\sigma_{HYPER/HYPO} = 0.8$. Therefore, the name affinity metric is defined as follows:

$$\boxed{NA(\mathrm{n(e_{ij}),n(e_{kl})}) = \begin{cases} 1 & \text{if } < \mathrm{n(e_{ij})}\ \mathit{SYN}\ \mathrm{n(e_{kl})} > \\ \sigma_{1R} * ... * \sigma_{(m-1)R} & \text{if } \mathrm{n(e_{ij})} \to {}^{m}\mathrm{n(e_{kl})} \\ 0 & \text{else} \end{cases}}$$

where $\mathrm{n(e_{ij})} \to {}^{m}\mathrm{n(e_{kl})}$ is a length of the path between e_{ij} and e_{kl} in the thesaurus and $m \geq 1$, σ_{nR} shows the weight of the nth relation in $\mathrm{n(e_{ij})} \to {}^{m}\mathrm{n(e_{kl})}$.

The semantic similarity is not sufficient to determine if two concepts are similar. We also need to compare their structures. The measure of the structural similarity of concepts is based on the calculation of their common properties and their common links with other concepts. Thus, to obtain the *Global Structural Similarity of Concepts (GSSC)* we need to measure the structural similarity of their properties and the structural similarity of their links with other concepts. These measures are respectively called *Structural Similarity of Concepts (SSC)* and *Adjacent Similarity of Concepts (ASC)*. The formulas are as follows:

$$\mathrm{GSSC(c_1,c_2)} = \frac{\mathrm{SSC(c_1,c_2) + ASC(c_1,c_2)}}{2} \qquad \mathrm{SSC(c_1,c_2)} = \frac{2 * (\text{Number of common properties in } c_1 \text{ and } c_2)}{\sum_{i=1}^{2} \text{Number of properties in } c_i}$$

$$\mathrm{ASC(c_1,c_2)} = \frac{2 * (\text{Number of commun adjacent concepts to } c_1 \text{ and } c_2)}{\sum_{i=1}^{2} \text{Number of adjacent concepts ato } c_i}$$

Process Models Similarity Measures. In this section we propose metrics to compare elements of process models i.e. of maps. Elements to compare are intentions, sections and maps themselves to evaluate the global similarity of maps.

We use two kinds of semantic similarity : the *Semantic Affinity of Intentions (SAI)* and the *Semantic Affinity of Sections (SAS)*. The *SAI* is used to measure the closeness of two intentions. This metric is based on the comparison of the two parameters composing the intention: verb and target by using the SYN relation. The *SAS* measures the closeness of two map sections. It is based on the measure of the *SAI* of its source intentions, the *SAI* of its target intentions and the application of the SYN relation between their strategies. The two formulas are defined as follows:

$$\mathrm{SAI}(i_i, i_j) = \begin{cases} 1 & \text{if } (i_i.\text{verb SYN } i_j.\text{verb}) \wedge (i_i.\text{target SYN } i_j.\text{target})) \\ 0 & \text{else} \end{cases}$$

$$\mathrm{SAS}(<i_i, i_j, s_{ij}>, <i_k, i_l, s_{kl}>) = \begin{cases} 1 & \text{if } \mathrm{SAI}(i_i, i_k) = 1 \wedge \mathrm{SAI}(i_j, i_l) = 1 \wedge s_{ij} \text{ SYN } s_{kl} \\ 0 & \text{else} \end{cases}$$

The structural similarity measures are needed to compare the structures of two maps and to identify their overlapping parts. We use two kinds of structural measures: the *Structural Similarity by Intentions (SSI)* and the *Structural Similarity by Sections (SSS)*. The *SSI* is used to measure the proportion of similar intentions in two maps. This is based on the calculation of the *SAI* of their intentions. The *SSS* allows us to measure the proportion of similar sections in two maps. Sometimes, we also need to compare the proportion of similar sections for a couple of intentions which exist in the two maps. For this we introduce the *Partial Structural Similarity (PSS)* metric. The three measures are defined as follows:

$$\mathrm{SIS}(m_1, m_2) = \frac{2 * \text{Number of similar intentions in } m_1 \text{ and } m_2}{\sum_{i=1}^{2} \text{Number of intentions in } m_i}$$

$$\mathrm{SSS}(m_1, m_2) = \frac{2 * \text{Number of similar sections in } m_1 \text{ and } m_2}{\sum_{i=1}^{2} \text{Number of sections in } m_i}$$

$$\mathrm{PSSS}(m_1 :< i_{1i}, i_{1j} >, m_2 :< i_{2k}, i_{2l} >) = \frac{2 * \text{Nb. of similar sctions between} < i_{1i}, i_{1j} > \text{ and} < i_{2k}, i_{2l} >}{\text{Nb. of sect. between} < i_{1i}, i_{1j} > + \text{ Nb. of sect. between} < i_{2k}, i_{2l} >}$$

m_1, m_2 : the maps; $m_1 :< i_{1i}, i_{1j} >$: a couple of intentions in the map m_1

4.2 Chunk Assembly by Integration

Figure 4 shows the process model corresponding to the assembly by integration strategy, the *integration map*. This map is a refinement of section <*Select a chunk, Assemble chunks, integration strategy*> of the APM (Figure 3).

The assembly process by integration, or the *integration process* for short, consists in identifying the common elements in the chunks product and process models and merging them. The maps of the these chunks must have some similar intentions and their product models must conceptualise the same objects of the real world by using similar concepts. As shown in Figure 4, the method engineer can start the assembly process by the integration of the process models followed by the integration of the product models or vice versa. At every moment he can navigate from the process models integration to the product models integration and vice versa.

Let us first consider the assembly of chunks process models, i.e. maps. It might be necessary to make some terminology adjustments of maps before their integration The mechanism of integration merges similar intentions that must have the same name. This is not necessarily the case in the initial chunks selected for assembly: intentions having the same semantics may have different names whereas semantically different intentions my be named exactly the same. The guideline of the *name unification strategy* helps to identify a couple of similar intentions requiring some name unification. The *SAI* (Sect. 4.1) measure is used to detect that the intentions are similar and then, the `RENAME_INTENTION` operator [12], [14] is recommended to unify their naming. Either directly or after having proceeded to the unification of names, the method engineer can move to the intention *Construct the integrated process model* following the *merge strategy* which recommends the use of the `MERGE_INTENTION` operator for each couple of similar intentions.

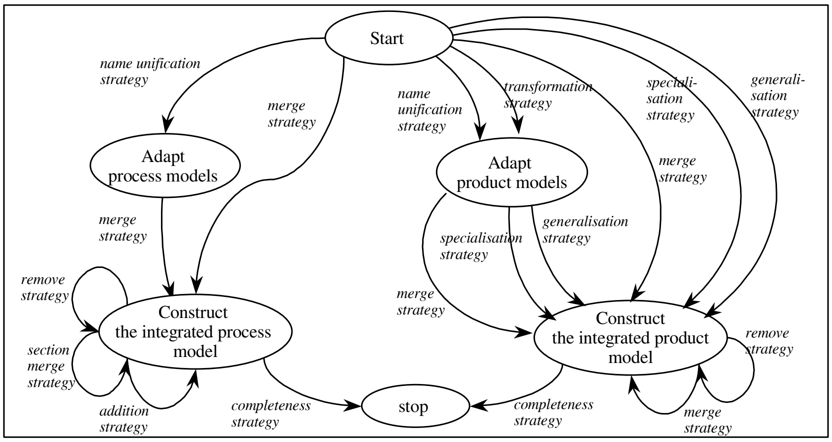

Fig. 4. Integration process map

The integration of product models is based on the identification of couples of similar concepts to be merged. Again, this might require naming revision or can be done directly. Two concepts to be merged must have the same semantics. In addition, if their structures are identical, they must have the same name. Vice-versa, if their structures are different, they must be named differently. For this reason, the product models integration may also be preceded by an adaptation step following the *name unification strategy* or the *transformation strategy*. The *name unification* strategy must be selected to solve the problem of naming ambiguity of concepts belonging to the different product models. The associated guideline uses the *NA* and *GSSC* measures (Sect. 4.1) to identify a couple of such concepts and proposes to rename one of them by applying the `RENAME_CONCEPT` operator. The *transformation* strategy must be selected when the same real world object is modelled differently in the two product models. For example, the object may be presented by a concept in one model and by a link between two concepts or by a structural property of another concept in the other model. The associated guideline helps to identify the couples of elements that need to be unified (concept and link, concept and property, or link and property) and to apply one of the product assembly operators `OBJECTIFY_LINK` or `OBJECTIFY_PROPERTY` according to the situation.

The same product integration strategies (*merge, generalisation* and *specialisation*) are possible to fulfil the intention *Construct the integrated product model* independently of the starting intention, the *Start* intention or the *Adapt product models* intention. The guidelines associated to the respective sections are identical. The *merge strategy* is applicable to merge concepts with similar semantics and similar structure. The corresponding guideline helps to identify a couple of similar concepts by applying the *NA* and *GSA* measures and to apply the product assembly operator MERGE_CONCEPT. The *generalisation strategy* shall be used when the two concepts have the same semantics but different structures: the GSA measure helps evaluating if the difference of their structures forbid their merging. The guideline associated to the corresponding sections proposes to generalise the two concepts into a new one by using the GENERALISE operator. The two initial concepts must have different names before their generalisation; therefore the name unification strategy shall be required first. Finally, the *specialisation strategy* is required when one concept represents a specialisation of the other concept. The associated guideline introduces a specialisation link between the two concepts by applying the SPECIALISE operator.

At any step of the integration process, it could be necessary to improve the current solution. The integration process map (Figure 4) proposes three strategies: *remove*, *addition* and *merge section*, to refine the integrated process model. The *remove strategy* deals with the need to remove elements in the integrated model. Many different reasons can justify such removals; for example to remove a useless or redundant guideline. The guideline associated to this section suggests the use of the REMOVE_SECTION operator to perform this operation. Some new guidelines can also be required to complete the integrated process model, particularly if the integrated product model integrates generalisation and/or specialisation of concepts. The integrated process model needs to be extended in these cases. The *addition strategy* helps doing so by applying the ADD_SECTION operator. Finally, the *merge section strategy* suggests to merge sections which are duplicates by applying the MERGE_SECTION operator.

Similarly, it can be necessary to improve the current version of the integrated product model. For example, the *remove strategy* allows to eliminate concepts, links or properties of the integrated product model by applying one of the operators REMOVE_CONCEPT or REMOVE_LINK or REMOVE_PROPERTY according to the situation at hand. To end the integration process the method engineer is invited to apply the quality rules and to verify the coherence and the completeness of the obtained product and process models following the *completeness strategy.*

4.3 Chunk Assembly by Association

In this section we consider the assembly of method chunks carried out following the *association strategy*. Figure 5 shows the process model corresponding to this assembly strategy represented by a map which is a refinement of the section <*Select a chunk, Assemble chunks, association strategy*> of the APM presented in Figure 3.

The assembly process by association, the *association process* for short, consists in connecting chunks such that the first one produces a product which is the source of the second chunk. Thus, the association process may consist in simply ordering chunks processes and relating chunks products to one another. The *association process* is simpler than the *integration process*. The association of product models is achieved by establishing links between concepts or adding elements connected to other concepts. The association of the process models consists in ordering the process activities provided by the two different models and possibly adding some new activity.

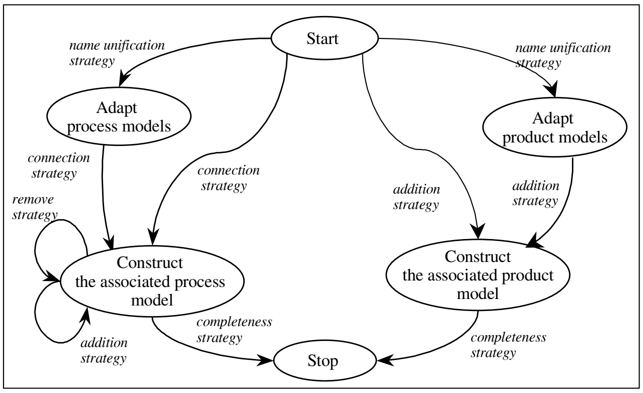

Fig. 5. The association process map

As in the case of the integration driven assembly, the association of chunks may also require the unification of their terminology. The *name unification strategy* is provided to unify names in maps and product models. Maps of chunks in this case should not have similar intentions as their product models should not contain similar concepts. The *SAI, NA* and *GSSC* (Sect. 4.1) measure must be applied to the suspected intentions and concepts and the RENAME operators must be applied if necessary.

If the two maps do not have any naming problems, the construction of the associated process model can start directly whereas the *Adapt process models* intention has to be fulfilled first in the reverse case. Then, the *connection strategy* is needed to carry out the association. The associated guidelines suggest a plan of action in three steps: (1) to determine the order in which the chunk processes must be executed; (2) to identify in the map of the first ordered chunk the intention that results in the product which is the source to the second chunk process, and (3) to merge this intention with the *Start* intention of the second chunk by applying the MERGE_INTENTION operator.

A similar set of strategies is proposed in Figure 5 to deal with the association of product models. The product models may by associated the *addition strategy* which advises to identify the concepts in the product models which can be connected by a link or by introducing an intermediary concept. These corresponding guidelines recommend the use of the product assembly operators ADD_LINK or ADD_CONCEPT depending of the situation at hand.

5 Application Example

In this section we illustrate the use of the method engineering process model with an example. We show how the method engineering map and its refined maps guide a method engineer step by step to construct a new method by retrieving and assembling method chunks.

Let us suppose that a method engineer has to construct a method supporting the elicitation of functional system requirements in a goal-driven manner, to conceptualise them using textual devices such as scenarios or use cases, to validate them in an animated fashion and finally to document them. According to the method engineering map presented in Figure 2, the first intention to achieve is to *Specify method requirements*. The *process driven strategy* looks adapted to the situation at hand as the requirements are expressed above in a process-oriented way. Assume that the application of this strategy leads to the requirements map presented in Figure 6.

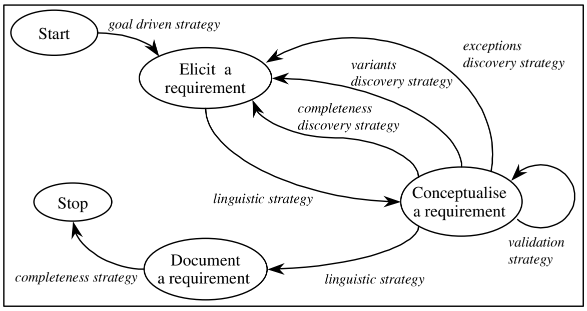

Fig. 6. The requirements map of the application example

Once the requirements for the new method have been elicited, the MEPM (Figure 2) suggests to *Assemble Chunks*. As the objective is to construct an entirely new method, the *from scratch assembly strategy* proposed in this method engineering map is chosen. The refined map of this assembly process (Figure 3) proposes to start with the selection of method chunks matching part or the totality of the requirements map. The guideline suggests to formulate queries to the method base in order to retrieve candidate method chunks. These queries give values to the different attributes of chunk interfaces and chunk descriptors [13], [14] as for example, design activity = *requirements engineering*, reuse intention = *discover functional system requirements*, situation = *problem description*.

Let's assume that method engineer selects the *L'Ecritoire* chunk as a candidate one. The process and product parts of this chunk are shown in Figure 7. This method chunk provides guidelines to discover functional system requirements expressed as goals and to conceptualise these requirements as scenarios describing how the system satisfies the achievement of these goals [17], [18]. Several different strategies are provided by the chunk to support goal elicitation, scenario writing and scenario conceptualisation. The method engineer wants to get a quantitative evaluation of the fit of *L'Ecritoire* to the requirements map. Therefore, he selects the *evaluation strategy* (Figure 3), which helps him to compare the map of the candidate chunk with the requirements map. The map similarity measures SAI, SAS, SSI and PSS (Sect. 4.1) are used. For example, owing to the SAI measure we detect that the intentions *Elicit a requirement* (requirements map) and *Elicit a goal* (*L'Ecritoire* map) are similar because they use the same verb and their targets *requirement* and *goal* are synonyms. The measure SSI, calculated as follows: *SSI (Requirements map, L'Ecritoire map) = (2*2 similar intentions) / (6 intentions in two maps) = 2/3*, shows that a large part of the requirements map is covered by the *L'Ecritoire* map. To validate this assumption, we search for similar sections by applying the SAS measure. For example, the SAS calculated as follows: *SAS (Requirements map: <Conceptualise a requirement, Elicit a requirement, variants discovery strategy>, L'Ecritoire* *map: <Conceptualise a scenario, Elicit a goal, alternative discovery strategy>) = 1*, shows that the concerned sections are similar. Next, for each couple of similar intentions we apply the PSS measure to verify if the strategies between these intentions are also similar. For instance, the *PSS (Requirements map: <Conceptualise a requirement, Elicit a requirement>, L'Ecritoire map: <Conceptualise a scenario, Elicit a goal>) = (2* 2 similar strategies) / (6 strategies) = 2/3* shows that the map of the *L'Ecritoire* matches a part of the requirements map. The degree of matching is satisfactory enough to select the *L'Ecritoire* chunk.

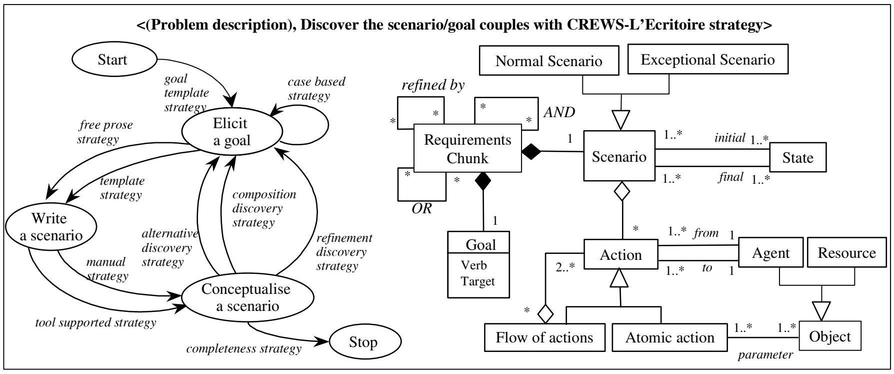

Fig. 7. The *L'Ecritoire* method chunk

The requirements coverage is not complete and the method engineer must continue the search. However, he knows the properties of the chunks he is looking for and can formulate precise queries. The required chunks must have the following values in their interfaces : situation = *goal* or *scenario*, intention = *to discover exceptional requirements (or goals).*

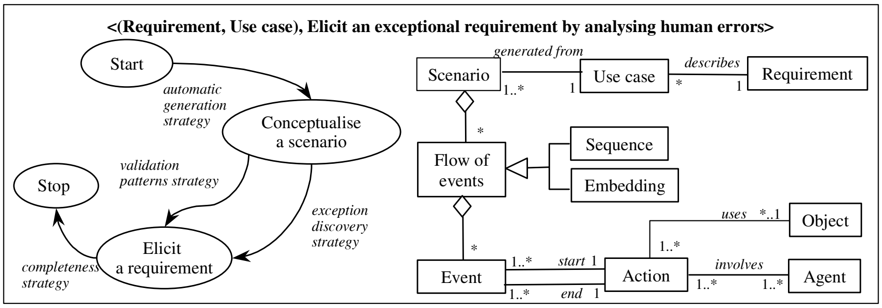

Fig. 8. The *SAVRE* method chunk

The *SAVRE* method chunk [23], [9] presented in Figure 9 is one of the chunks retrieved by the query. This chunk provides guidelines to discover exceptions in the functioning of a system under design caused by human errors. It generates scenarios corresponding to the system requirements and identifies, through an analysis of these scenarios, possible exceptions caused by human errors (*exception discovery strategy*).

The chunk also includes validation patterns to validate the requirements (*validation patterns strategy*).

The matching measures convinced the method engineer to make the decision to assemble the two selected method chunks, thus to move in the APM (Figure 3) to *Assemble chunks.* The two chunks have the same broad objective, to discover system requirements, and their process and product models overlap (they contain similar intentions and concepts). Thus, the *integration strategy* to assemble these chunks is adequate.

Following the integration map shown in Figure 4, the method engineer understands that he first needs to adapt the product and process models of the two chunks. It is only after the necessary terminological adaptations that he will be able to proceed to their integration. As an example, he selects the *name unification strategy* in the integration map (Figure 4) and changes the name of the intention *Elicit a requirement* in the *SAVRE* map into *Elicit a goal* by applying the `RENAME_INTENTION` operator. Then, he progresses to the construction of the integrated process model with the *merge strategy* to integrate the two maps. He applies the `MERGE_INTENTION` operator on the couples of identical intentions. The merged intentions are represented in grey in Figure 10. By selecting the *addition strategy* in the integration map he adds the *transformation strategy* to the integrated map. This new strategy permits the coupling of the two types of scenarios (the ones in *L'Ecritoire* and the ones in *SAVRE*) in the same integrated product and the transformation of *L'Ecritoire* scenarios into *SAVRE* scenarios and vice versa.

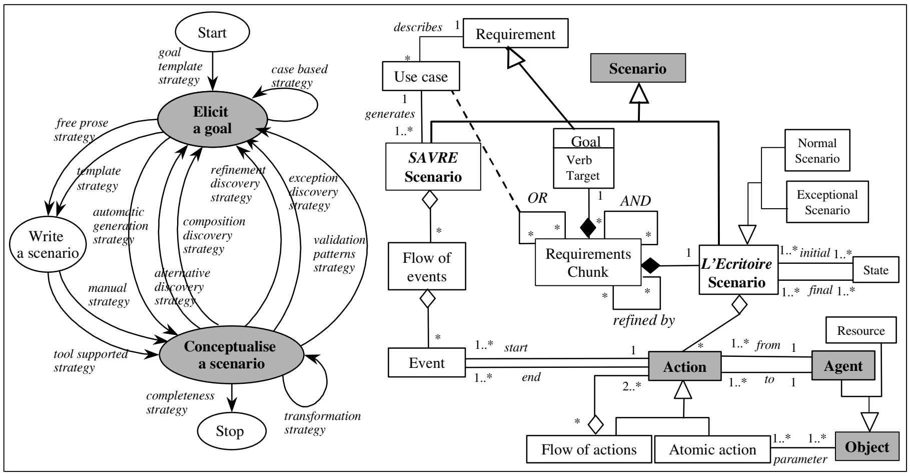

Fig. 9. The process and the product models of the integrated method chunk

The integration of the two product models also requires some adaptations. For example, the two product models contain the *scenario* concept. The two *scenarios* have the same semantics, but their structures are different. Thus, the method engineer renames the *scenario* concept in the *L'Ecritoire* into *L'Ecritoire scenario* and in the *SAVRE* into *SAVRE scenario.* Then he selects the *generalisation strategy* in the integration map to integrate the two scenario concepts by applying the `GENERALISE` operator. The notion of the *Agent* is the same in the two product models. Thus, the *merge strategy* can be selected to help applying the MERGE_CONCEPT operator on these two concepts. The result of the integration of *two* method chunks is illustrated in Figure 10.

The requirements coverage is still not completed and the method engineer continues the search for chunks that can fill in the gap between the requirements map and the integrated chunk. There is a need for validating the requirements. Thus, the method engineer formulates a new query asking for chunks with the intention to *validate the requirements* in their interface. Among the retrieved method chunks, the method engineer retrieves the *Albert* chunk [7] presented in Figure 11. This chunk proposes guidelines to validate requirements in an animated manner. It can transform scenarios describing requirements into an *Albert* specification and then, supports the animation of these scenarios by activating the tool called *Animator*.

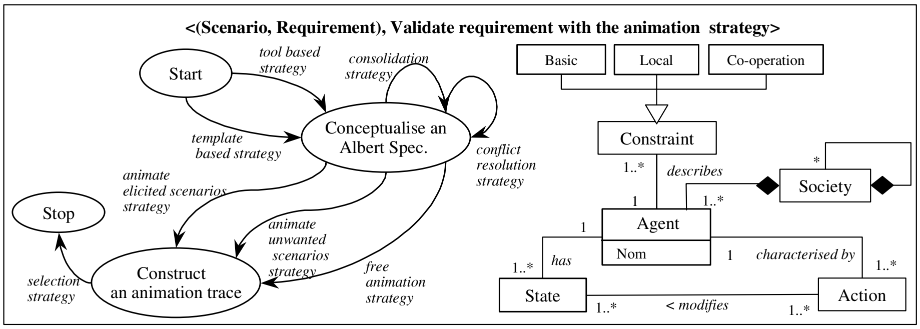

Fig. 10. The *Albert* method chunk

The maps of the two chunks to assemble do not have similar intentions. Thus, there is no need to adapt the maps before their association. The method engineer selects the *connection strategy* in the association map (Figure 5) to the construction of the associated process model. Following the associated guideline, he identifies that the achievement of the intention *Conceptualise a scenario* in the integrated map constructs a product (a scenario) which is a source product for the *Albert* chunk. The operator MERGE_INTENTION is used on the intention *Conceptualise a scenario* and the *Start* intention of the *Albert* map. Some refinements are necessary on the associated map. For example, it seems reasonable to forbid a progression from the intention *Conceptualise a scenario* to *Stop*. By selecting the *remove strategy* in the association process map (Figure 5) the method engineer applies the operator REMOVE_SECTION on this section.

The construction of the associated product model consists in the adaptation of the *Agent*, *State* and *Action* concepts and addition of the links between the corresponding concepts. The end result is shown in Figure 12 (only the final map).

In a similar manner the selection of additional chunks to cover the entire requirements map and their assembly with the current integrated chunk will continue till the completeness strategy ensures that the result is satisfactory enough to stop the assembly process.

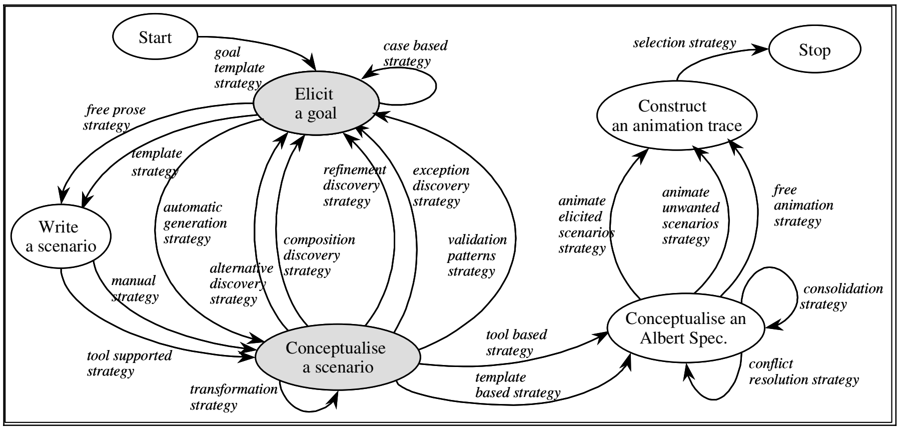
Start
goal template strategy
case based strategy
Elicit a goal
selection strategy
Stop
Construct an animation trace
free prose strategy
template strategy
automatic generation strategy
refinement discovery strategy
exception discovery strategy
animate elicited scenarios strategy
animate unwanted scenarios strategy
free animation strategy
Write a scenario
composition discovery strategy
validation patterns strategy
alternative discovery strategy
manual strategy
consolidation strategy
Conceptualise an Albert Spec.
tool based strategy
tool supported strategy
Conceptualise a scenario
template based strategy
conflict resolution strategy
transformation strategy

Fig. 11. The end result of the chunk assembly

6 Conclusion

In this paper we look at situational method engineering from a process perspective and propose two embedded generic models to support:

– method construction, and
– method chunk assembly.

Both are concerned with engineering methods matching a set of requirements through a method chunk assembly technique. The former deals with assembly ‘in the large’ whereas the latter offer solutions ‘in the small’.

The process models are represented as maps with associated guidelines. This allows us to offer flexibility to the method engineer for carrying out the engineering activity. Besides, guidelines provide a strong methodological support, thanks to some formally defined techniques. Metrics to evaluate the distance between two method chunks and a set of operators to perform the assembly tasks are the two most important techniques.

The approach is currently used in a professional environment in the context of a rather large project (§10 millions). Results are encouraging, the experience is positive, even if it highlights the need for improvements among which is a software environment to support the process.

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. Bianco G., V. De Antonellis, S. Castano, M. Melchiori, *A Markov Random Field Approach for Querying and Reconciling Heterogeneous Databases*. Proc. of the 10th Int. Workshop on Database and Expert Systems Applications (DEXA’99), Florence, Italy, September 1999.

3. Brinkkemper S., M. Saeki, F. Harmsen, *Assembly Techniques for Method Engineering*. Proc. of the 10th Conf. on Advanced Information Systems Engineering, Pisa Italy, 1998.
4. Castano S., V. De Antonellis, *A Constructive Approach to Reuse of Conceptual Components*. Proc. of Advances in Software Reuse: Selected Papers from the Second International Workshop on Software Reusability, Lucca, Italy, March 24-26, 1993.
5. Harmsen A.F., S. Brinkkemper, H. Oei, *Situational Method Engineering for Information System Projects*. Proc. of the IFIP WG8.1 Working Conference CRIS'94, pp. 169-194, North-Holland, Amsterdam, 1994.
6. Harmsen A.F., *Situational Method Engineering*. Moret Ernst & Young , 1997.
7. Heymans P., E. Dubois, *Scenario-Based Techniques for Supporting the Elaboration and the Validation of Formal Requirements*. Requirements Engineering Journal, 3 (3-4), 1998.
8. Jilani L.L., R. Mili, A. Mili*, Approximate Component Retrieval : An Academic Exercise or a Practical Concern ?* Proceedings of the 8th Workshop on Istitutionalising Software Reuse (WISR8), Columbus, Ohio, March 1997.
9. Maiden N.A.M., *CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements.* Journal of Automated Software Engineering, 1998.
10. 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.* Proc. 5th Int. Conf. on Software Process (ICSP'98), Chicago, Illinois, USA, 14-17 June 1998.
11. Punter H.T., K. Lemmen, *The MEMA model : Towards a new approach for Method Engineering*. Information and Software Technology, 38(4), pp.295-305, 1996.
12. Ralyté J., C. Rolland, V. Plihon, *Method Enhancement by Scenario Based Techniques*. Proc. of the 11th Conf. on Advanced Information Systems Engineering CAISE'99, Heidelberg, Germany, 1999.
13. Ralyté J., *Reusing Scenario Based Approaches in Requirement Engineering Methods: CREWS Method Base*. Proc. of the First Int. Workshop on the RE Process - Innovative Techniques, Models, Tools to support the RE Process, Florence, Italy, September 1999.
14. Ralyté J., *Ingénierie des méthodes par assemblage de composants*. Thèse de doctorat en informatique de l'Université Paris 1. Janvier, 2001.
15. 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.
16. Rolland C., V. Plihon, J. Ralyté, *Specifying the reuse context of scenario method chunks*. Proc. of the 10th Conf. on Advanced Information Systems Engineering, Pisa Italy, 1998.
17. Rolland C., C. Souveyet, C. Ben Achour, *Guiding Goal Modelling Using Scenarios*. IEEE Transactions on Software Engineering, 24 (12), 1055-1071, Dec. 1998.
18. Rolland C., C. Ben Achour, *Guiding the construction of textual use case specifications*. Data & Knowledge Engineering Journal, 25(1), pp. 125-160, March 1998.
19. Rolland C., N. Prakash, A. Benjamen, *A multi-model view of process modelling*. Requirements Engineering Journal, p. 169-187,1999.
20. 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.
21. 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.
22. Song X., *A Framework for Understanding the Integration of Design Methodologies.* In: ACM SIGSOFT Software Engineering Notes, 20 (1), pp. 46-54, 1995.
23. Sutcliffe A.G., N.A.M. Maiden, S. Minocha, D. Manuel, *Supporting Scenario-based Requirements* Engineering. IEEE Transactions on Software Engineering, 24 (12), 1998.