Requirements Eng (2004) 9: 1–15
DOI 10.1007/s00766-003-0168-y

ORIGINAL ARTICLE

Colette Rolland · Camille Salinesi · Anne Etien

Eliciting gaps in requirements change

Published online: 18 June 2003

Abstract We consider requirements change due to system evolution which results from contextual forces such as the decision to standardise practices across subsidiaries of a company. Our experience with the financial branch of the French Renault group is that eliciting change requirements poses its own specific problems. We propose to model change as a set of gaps between the requirements specification of the current and the future system. Our approach is to define a generic typology of gaps to facilitate a precise definition of change requirements. It adopts a goal oriented requirements specification and shows how to customise the generic gap typology to this specific requirements representation formalism. The paper presents the approach to elicit gaps and illustrates it with the Renault case study.

Keywords System evolution · System change · Change gap · Change primitive · Gap meta-model

1 Introduction

System evolution is a fact of industrial life. In order to survive in a more and more competitive environment, organisations undergo frequent changes that imply changes of their software-based systems. As a consequence of the fast speed of organisational change, system evolution costs are far higher than initial development costs. Lientz and Swanson [1] and Nosek and Palvia [2] show for example that more than 50% of total life cycle cost is incurred after initial development has been done. Handling system evolution has therefore been an important issue, sufficiently recognised worldwide to be the subject of the series of IPSE international workshops [3]. Whereas a large number of approaches deal with the evolution of the software model and its impact on running instances, we are concentrating in this paper on the *system requirements level*. The former focus on the identification of primitive modifications, evolution patterns and change propagation for different types of software models such as (a) database schemas [4, 5, 6], (b) workflow models [7, 8, 9] and (c) software process models [10, 11, 12]. Instead, in the latter, following Lehman's view that 'software evolution is driven by the need to maintain user satisfaction' [13], we concentrate on *requirements change*.

Our position is that a prerequisite to performing software change is to understand the way organisational changes impact system requirements. Just as for systems developed from scratch where requirements elicitation provides a basis for subsequent software design and implementation, changes in design and implementation are rooted in changed requirements. *Eliciting these change requirements* is the concern of this paper.

Requirements engineering is intrinsically concerned with change to such an extent that it has been defined in [14] as the process of establishing the vision for change in the organisational context. The focus has been naturally placed on modelling the vision and the constraints imposed by the context, on what Jackson calls the optative properties [15]. In his survey of researches conducted in the field of requirements engineering, Lamsweerde [16] demonstrates the key role played by goals to concretise the vision [14] and to refine high-level strategic vision into low-level system constraints [17]. Despite our conformity with the goal-oriented view of requirements modelling, our concern is on *change requirements* for a specific type of evolution that we refer to as system adaptation.

The nature of change requirements varies with the nature of system evolution. There are many kinds of system evolution and we focus on those that are caused by changes in the organisational context in which a legacy software system is now to operate. We will refer to this type of system evolution as *system adaptation*. Our experience with European companies shows that such

C. Rolland (✉) · C. Salinesi · A. Etien
CRI, Université Paris 1, Sorbonne,
90 rue de Tolbiac, 75013 Paris, France
E-mail: rolland@univ-paris1.fr

changes of organisational context are currently frequent. They typically occur when mergers/take-overs, globalisation, standardisation of practices across branches of a company etc. occur. Several legacy software systems are already running when such events take place. In such a context, it is out of the question to develop a new system from scratch but it is possible to integrate these legacy systems or to select one of these for adaptation and uniform deployment across the organisation.

Typically, system adaptation is bounded by the following constraints:

– No large-scale deviations in the selected software system.
– Compliance with some of the functionality not found in the selected system but provided by others.
– Provision of functionality for handling new business opportunities that are now recognised to be important.

From the foregoing it seems to us that the adaptation process should have two characteristics:

1. It has to be put into the larger context of organisational change and, therefore, must concentrate first on the *requirements for system change*. Thereafter, traceability links can be used to propagate the change requirements into actual software changes.
2. It should be driven by *gaps* which identify what has to be changed/adapted to the new situation. In this change context, it is not so much the representation of the future situation that is important (as it is fixed to a large extent) as the difference from the current situation. If gaps remain implicit, it is difficult to identify what has to be changed. Explicit gap representation seems to us, therefore, crucial to expressing change requirements.

We adopt the change-handling view in which change creates a movement from an existing situation captured in the *As-Is model* to a new one captured in the *To-Be model* [18]. According to Jackson [15], the *As-Is* models describe indicative properties whereas the *To-Be* models describe optative properties. In the approach presented in this paper the As-Is and To-Be models are *requirements models* expressed as *maps* (Fig. 1). A map [19, 20] is a goal-oriented requirements representation formalism that allows the representation of a set of requirements as a non-deterministic ordering of *intentions* and of *strategies* to achieve them. By avoiding unnecessary details, maps help in focusing attention on what is to be achieved (the intentions) and the ways required to achieve them (the strategies). The As-Is map abstracts from the technical details of the implemented system to focus on the business goals and associated strategies to attain them that are supported by the system. The To-Be map expresses the intentions and their associated strategies that the organisation wants to achieve in the future.

The approach departs from Jarke's view by modelling the changes to be made as a collection of *gaps* between

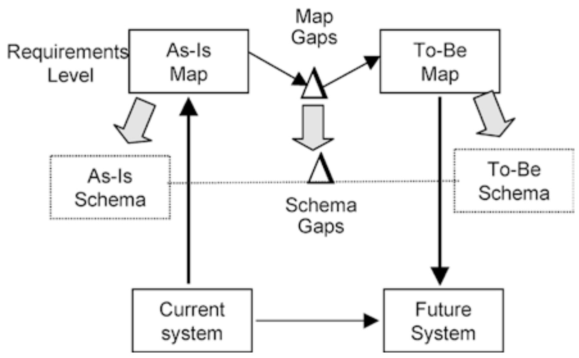

Fig. 1 The gap-driven approach

the *As-Is* and the *To-Be* maps. A gap expresses a difference between the initial situation, the As-Is map and the future situation, the To-Be map. Intuitively gaps might be related to operators that cause transformations of the As-Is map into the To-Be map. This is the view developed in this paper. Map gaps express the requirements for change implied by the organisational business expectations and the needs for the future. An expression of gaps constitutes the requirements specification for change, the *change requirements*.

In the adaptation process it is essential to describe the gaps precisely since they are the basis of the actual change of the software system. As suggested in Fig. 1, map gaps might be propagated into schema gaps describing the modifications in system functionality implied by the change requirements. The actual adaptation of the system is based on the schema gaps whereas map gaps provide the business rationale for these schema gaps. However, in this paper, our interest is with the requirements level only.

In the proposed approach to elicit change requirements, gaps are precisely defined by associating each gap to a predefined type of change. The paper proposes a generic *gap typology* and its customisation to expressing gaps between maps. The *map gap typology* is a set of operators associated to the map representation system.

The approach for software system adaptation presented in this paper was developed to handle the standardisation of practices in the Financial branch, DIAC, of the Renault motor company. The current software system, besides providing other financial services, deals with granting credit to Renault customers. A number of such systems are running in DIAC subsidiaries located in different countries, and it was required to standardise these across Europe. The Spanish software system was selected for adaptation and deployment in France, Spain, Portugal and Germany.

The adapted software system, called FUSE, must comply with the functionalities available in the French software system and meet all the financial regulations in the different countries. There are new business needs as well as (a) diversification of the sales channels to include, for example, sales by the internet in addition to regular vendors, (b) inclusion of additional financial services, for example, offering personal loans in addition to car loans, and (c) planning for the same software system to be used across Europe.

In the next section we present the generic *gap typology*. In Sect. 3 we provide an overview of the map formalism to represent system requirements and present the customisation of the gap typology for maps, i.e., the *map gap typology*. Section 4 outlines the process of eliciting gaps. Section 5 illustrates this gap elicitation process with examples extracted from the DIAC project.

2 The gap typology

Our proposal is to model the change movement from the current situation to the future situation as *gaps* between the *As-Is map* and the *To-Be map*. Intuitively a gap expresses a difference between the As-Is map and the To-Be map such as the deletion or addition of an As-Is element in the To-Be map. Gaps are related to *operators* which transform elements of maps. In order to facilitate the elicitation of gaps in the adaptation process, we defined a *typology of map gaps*, i.e., a set of predefined gap types related to operators acting on maps. For instance, as 'intention' is a type of element of a map, 'remove intention' is part of the map gap typology.

A basic question is which requirements the set of operators should fulfil. This question has been addressed in object-oriented schema evolution by Banerjee [6], who introduces two important properties: *completeness* and *correctness*. A set of operators is considered to be complete if it subsumes every possible schema evolution; it is correct if the execution of any operator does not result in an incorrect schema. In [6] the correctness of the schema is defined by a set of schema invariants which are conditions over the schema. These properties seem to have been accepted as a standard (e.g., Casati [7] and Kradolfer [21]) and we adopt them for the map gap typology.

The set of gap types is *complete* in the sense that it subsumes every possible type of map change. Thus, every actual gap between two maps is expressed as an instance of a gap type. The map gap typology ensures the *correctness* property: map *invariants* are maintained by the execution of any operator of the map gap typology.

2.1 Towards a generic gap typology

Obviously a relevant map gap typology could have been defined in an ad hoc sort of manner. However, besides the fact that this might be error prone, the resulting typology would be dependent on the specific requirements specification formalism used in this paper, i.e. the map formalism. To overcome these difficulties, we search for a *generic gap typology*, i.e. a typology that is independent of the formalism used to express the As-Is and the To-Be models. The map gap typology will, then, be an instance of the generic gap typology. Other gap typologies such as an object-oriented gap typology or a goal gap typology could be easily generated as well.

There are some advantages of proceeding in this way:

1. The generic typology serves as a guide to define the specific typology: the latter is just an instance of the former.
2. The completeness of the specific typology is subsumed by the completeness of the gap typology.
3. Specific typologies are consistent with each other as they are generated from the same mould: this is important when several typologies are used in the same method.

The generic gap typology will take the form of a set of operators applicable to generic elements that compose any model. To achieve this, it is necessary to abstract from the specificity of a given model such as the map model to generalise model elements and their relationships. Meta-modelling is known as a means to do so. Thus, in order to build the generic gap typology, we first develop a *meta-model*, i.e. a model of models.

2.2 A meta-model for defining the generic gap typology

A number of attempts have been made to make explicit the elements that compose any model, i.e. to define meta-models [22, 23, 24, 25, 26]. There are different meta-models depending on the meta-modelling purpose. For example, IRDS [22] is a standard to facilitate the evolution of model representation in CASE tools, Prakash [26] aims at a formal definition of a method and Marttiin [23] searches for a generic repository structure of meta-Case environments. The meta-model we developed is targeted to the identification of key significant transformations that can occur in a model.

This meta-model is drawn in Fig. 2 using UML notations. This figure shows that any model is made of *Elements,* every element having a *Name*, and is characterised by a set of *Property*. In the E/R model, for example, *Entity type, Attribute* and *Relationship type* as well as the *Is-A* relationship are *elements*. *Domain* is a *property* of *Attribute*.

In the meta-model there are two orthogonal classifications of *Elements*. The first classification makes the distinction between *Simple* and *Compound Elements*. *Compound elements* are decomposable into fine-grained ones that can be simple or compound elements whereas *Simple Elements* are not decomposable into other *Elements*. The second classification is a partition of elements into *Link* and *NotLink*. An element of the type *Link* is a connector between two elements, one being the *Source* and the other the *Target*. Elements, which are not links, are referred to as *NotLink*. In the E/R model an *Entity type* is a compound element made of *Attributes* which are simple elements. An *Is-A relationship* of the E/R model is a *Link*: it connects a source *Entity type* to a target *Entity type*. Vice versa, an *Entity type* is *NotLink*.

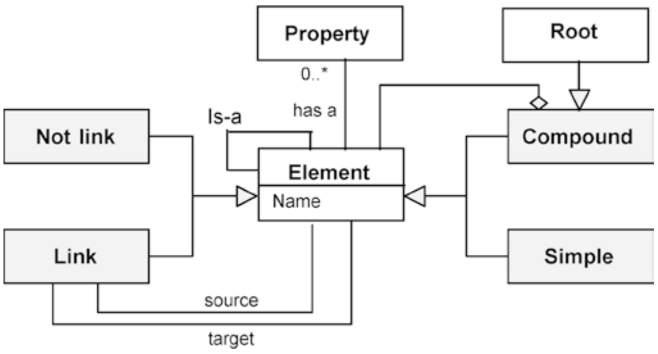

Fig. 2 The meta-model for gap typology definition

Figure 2 shows that an element *is-a* another element, i.e. might inherit from another element Finally, any model is a compound element which can be reduced to the *root* element.

2.3 The generic gap typology

The *generic gap typology* is composed of a set of *operators* applicable to *Element*. Each operator identifies a type of change that can be performed on an As-Is model. The operator identifies the difference between the As-Is model and the To-Be model. For example, as *Rename* is an operator (see Table 1), *Rename Element* will be a change that characterises the transformation of an As-Is element in the To-Be model.

The generic gap typology identifies three major types of change: *naming* changes, *element* changes and *structural* changes.

– *Naming changes* are defined with the *Rename* operator. They only affect the way organisations want to refer to an element. Naming is dealing with hyponyms, synonyms and the like.
– *Element changes* affect elements and are circumscribed to the elements themselves: adding an *attribute* to an *entity type* is an example of such localised change. Table 1 proposes four operators to specify element changes, namely *Modify*, *Give*, *Withdraw* and *Retype*.
– *Structural changes* are the most important as they correspond to a modification of the set of elements which composes the model. There are nine operators to specify structural changes in Table 1: *ChangeOrigin*, *AddComponent*, *MoveComponent*, *RemoveComponent*, *Replace*, *Split*, *Merge*, *Add* and *Remove*. For example, adding or removing *Relationship types* and *Entity types* in an *As-Is* E/R schema to form the *To-Be* schema is a structural change.

Table 1 sums up the *generic gap typology* composed of 14 operators that we identified on *Element*.
In conformance with [6] we believe that the proposed collection of change operators is complete, since:

– Any model can be generated from the root element by a finite sequence of *Add*, *AddComponent* and *Give* operations.
– Any existing model can be reduced to the root element by a finite sequence of *Remove*, *RemoveComponent* and *Withdraw* operations.

3 The map gap typology

The *map gap typology* is an instance of the generic gap typology. As the generic one, the map gap typology is based on a set of operators adapted to the specific elements that compose a map. In order to define the map gap typology, we proceed in two steps to:

1. Instantiate the meta-model for identifying the specific map elements and their properties.
2. Customise the generic operators for each specific element of the map.

3.1 Map as an instance of the meta-model

Figure 3 shows the instantiation of the meta-model for maps. Figure 3 shows the key concepts of a map and their type as meta-model elements. These are as follows:

Table 1 Meta-model elements and related operators

Object	Operator	Description
Element	*Rename*	Change the name of the element in the To-Be model
	Add, Remove	Add/Remove an element of the As-Is in the To-Be model
	Merge	Two separate As-Is elements become one in the To-Be model
	Split	One As-Is element decomposes into two To-Be elements
	Replace	An As-Is element is replaced by a different To-Be one
Link	*ChangeOrigin*	The source or target of the link is changed
Compound	*AddComponentt*	A component is added in the To-Be element
	RemoveComponent	An As-Is component is removed in the To-Be element
	MoveComponent	A component is repositioned in the structure of the To-Be element
Property	*Give*	Add a property to the To-Be element
	Withdraw	Remove an As-Is property in the To-Be element
	Modify	Change the property of the To-Be element
	Retype	The As-Is and To-Be elements have different types

- A *Map* is a *compound element* composed of *Sections*, each section being an aggregation of two types of *Intentions*, the *Source Intention* and the *Target Intention* together with a *Strategy*. Therefore, there are three key elements in a map: *Intention*, *Strategy* and *Section*, which we describe in turn.
- An *Intention* is a goal that can be achieved by the performance of an activity (automated/semi-automated or manual). For example, *Make Loan Demand* is an intention to formulate a request for loan. Similarly, *Make Loan Decision* is another intention. We postulate that each map has two special intentions, *Start* and *Stop*, to begin and end the map respectively. We use a linguistic approach to provide a template to formulate an intention [27]. An intention is expressed as a clause with a main verb and several parameters, where each parameter plays a different role with respect to the verb. A detailed description of the intention structure can be found in [28].

An example of an intention is the following:

$$\text{Collect}_{\text{verb}}(\text{requests})_{\text{target}}(\text{from customers})_{\text{source}}$$
$$(\text{for subcontractor vendors})_{\text{beneficiary}}$$

In Fig. 3, the *Verb*, *Target* and *Parameters* are shown as three *properties* of the *Intention element*. It is also shown that *Intention* is a *simple element* as it cannot be decomposed into other elements. Besides an Intention is a *NotLink* type of *Element*.

- A *Strategy* is an approach, a manner to achieve an intention. In our example, let it be required that demands can be made on the Internet. This is a way of achieving our *Make Loan Demand* intention: '*by Internet'* is a strategy. In Fig. 3 a *Strategy* is shown as a *Simple* element of the type *Link*. As a link, a strategy has a source which is the *Source Intention* and a target which is the *Target Intention*.
- A *Section* is an aggregation of the *Source Intention*, the *Target Intention*, and a *Strategy*. A section expresses the strategy using which, starting from a source intention, the target intention can be achieved. For example, the aggregation of the source intention *Start*, the target intention *Make Loan Demand* and the *by Internet* strategy defines a section *< Start, Make Loan Demand, by Internet >*. Here, the *by Internet* strategy characterises the flow from the source intention *Start* to the target intention *Make Loan Demand* and the way the target can be achieved.
- A *Section* can be seen as a transition from an initial state attained by the achievement of the source intention to a final state resulting from the achievement of the target intention through the execution of the business rule associated to the section strategy. These aspects are captured by the three *properties* attached to the *Section* element in Fig. 3: the *pre-condition* (characterising the initial state), the *post-condition* (reflecting the final state) and the *business rule*. For instance, the pre-condition of the section *< Start, Make Loan Demand, by Internet >* is that the site for e-loaning is operational; the post-condition is that the demand has been registered as the result of following the electronic procedure for expressing a loan demand.

By definition, sections are connected one another. This occurs:

- When an intention can be achieved only if another one has been achieved. This establishes a precedence/succedence relationship between sections that is called a *path*. In a path the target intention of the preceding section is the source intention of the succeeding section. In our loan example, in addition to the intention to *Make Loan Demand by Internet* let there be another intention to *Make Loan Decision by Risk Analysis strategy*. Evidently, this intention can be fulfilled after a demand for loan is made … Thus, there is a path relationship between the section *< Make Loan Demand, Make Loan Decision, Risk Analysis strategy>* and the section *< Start, Make Loan Demand, by Internet >*: *Make Loan Demand* is the target of one section and the source of the other.
- When a given intention can be performed using different strategies. This is represented in the map by several sections between a pair of intentions. Such a map topology is called a *multi-thread*. For example, let it be required that decisions can be made manually or automatically. Thus we have two ways *to Make Loan Decision, Make Loan Decision by Risk Analysis* and *Make Loan Decision Manually*, both of which (by an extension of the discussion above) have the same

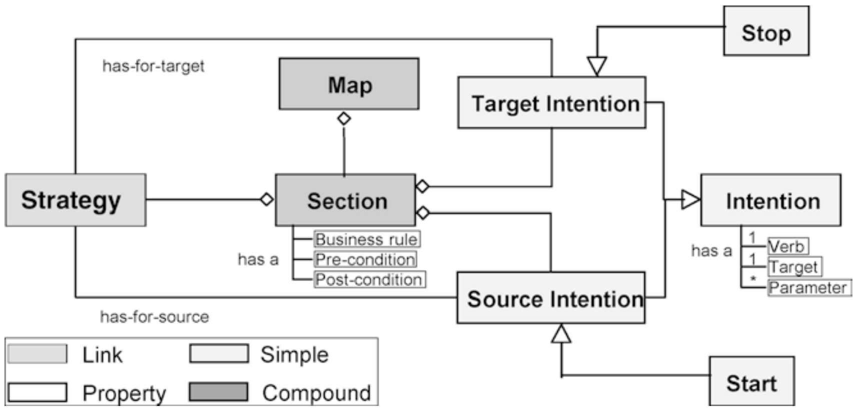

Fig. 3 Instantiating the meta-model for maps

source intention, *Make Loan Demand* and the same target intention *Make Loan Decision*. Then the two sections <*Make Loan Demand, Make Loan Decision, By Risk Analysis*> and <*Make Loan Demand, Make Loan Decision, Manually*> are in a thread relationship with one another because they represent two different ways of achieving *Make Loan Decision* from *Make Loan Demand*.

In general, a map from its *Start* to its *Stop* intentions is a multi-path and may contain multi-threads.

We represent each map as a directed graph from *Start* to *Stop*. In this graph, intentions are represented as nodes and strategies as edges between these. The graph is directed because the strategy shows the flow from the source to the target intention. As an example consider the map shown in Fig. 4 which contains six sections MS0 to MS5.

MS1 and MS2 constitute a multi-thread. There are multi-paths from *Start* to *Stop*: MS0, MS4, MS3, MS1, MS5 is one example; MS0, MS4, MS2, MS5 is another one.

Finally, let us mention that it is possible to *refine* a section of a map at level *i* into an entire map at a lower level *i*+1 to view an intention together with its strategy as a complex graph of intentions and their associated strategies. Refinement as defined here is an abstraction mechanism by which a complex assembly of sections at level *i*+1 is viewed as a unique section at level *i*. Since refinement results in a map, it produces multi-path/multi-thread structures at level *i*+1. Refinement makes it possible to progressively move from the highest business requirements into finer ones that can be operationalised in software functionality. We will see in Sect. 4 how the refinement mechanism helps in considering gaps at different levels of detail.

3.2 Map invariants

In order to ensure the correctness property of the map gap typology, we need to define what a *correct* map is. This is achieved by adding to the structural definition of a map presented in the above section a set of properties called *invariants*. The invariants must hold in any quiescent state of a map, that is, before and after any execution of a change operator to the As-Is map resulting in a new state of the To-Be map. The invariants guide the definition of the semantics of every meaningful map change, by ensuring that the change operator does not leave the To-Be map in an incorrect state, that is, a state which violates any invariant. We have been able to identify the three following invariants of a map:

- **I1**. Any map has one and only one intention which is the target of no strategy; that is, the *Start* intention.
- **I2**. Any map has one and only one intention which is the source of no strategy; that is, the *Stop* intention.

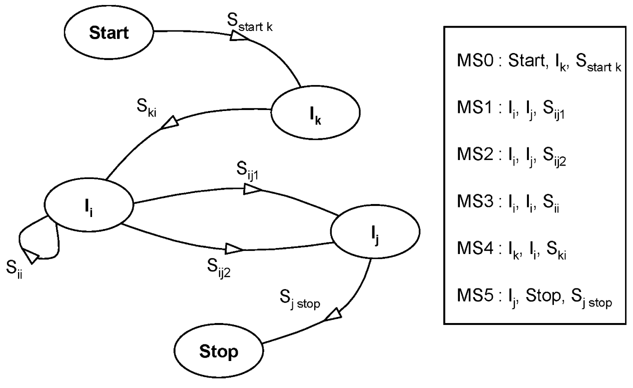
Start
$S_{start\ k}$
I_k
S_{ki}
S_{ij1}
I_i
I_j
S_{ii}
S_{ij2}
$S_{j\ stop}$
Stop
MS0 : Start, I_k, $S_{start\ k}$
MS1 : I_i, I_j, S_{ij1}
MS2 : I_i, I_j, S_{ij2}
MS3 : I_i, I_i, S_{ii}
MS4 : I_k, I_i, S_{ki}
MS5 : I_j, Stop, $S_{j\ stop}$

Fig. 4 The map as a graph

- **I3**. Any intention in a map must be *quasi-live*. An intention is said *quasi-live* if it can be achieved at least once, i.e. if there exists a path from *Start* to this intention. In Fig. 4, intention I_j is *quasi-live* because it exists at least one path from *Start* (for instance MS0, MS4, MS1) to achieve it.

I1, I2 and I3 have several corollaries:

- **C1**. Maps are connected graphs; there is no isolated intention or strategy.
- **C2**. Any intention in a map is the source of a strategy except the *Stop* intention.
- **C3**. Any intention in a map is the target of a strategy except the *Start* intention.
- **C4**. There is always a path from *Start* to *Stop*.
- **C5**. Any section belongs to a path between *Start* and *Stop*.

3.3 Map gap typology

Having determined the elements that compose a map, their types and properties, we can now identify the map change types, i.e. the *map gap types*. They correspond to the instantiation of the set of the 14 generic operators (Table 1) for the specific map elements. Table 2 sums up the 25 types of map gaps applicable to maps.

Table 2 comprises 11 lines corresponding to 11 of the 14 generic types of gaps identified in Table 1. The three missing operators, namely *AddComponent*, *RemoveComponent* and *MoveComponent*, have not been introduced as it does not make sense to apply them to the *Section* element. Indeed the structure of a section is immutable and is composed of a source intention, a target intention and a strategy. As a map comprises three elements: Intention, Strategy and Section, Table 2 is organised in three columns. Each generic operator is potentially applicable to each of the three elements. For example, *Add,* which is a generic operator, is applicable to *Intention*, *Strategy* and *Section*. This identifies three map gap types, namely *AddIntention, AddStrategy and AddSection*.

Table 2 Map gap types

Operator	Intention	Strategy	Section
Rename	*RenameIntention*	*RenameStrategy*	*RenameSection*
Add	*AddIntention*	*AddStrategy*	*AddSection*
Remove	*RemoveIntention*	*RemoveStrategy*	*RemoveSection*
Merge	*MergeIntention*	*MergeStrategy*	*MergeSection*
Split	*SplitIntention*	*SplitStrategy*	*SplitSection*
Replace	*ReplaceIntention*	N/A (not applicable).	N/A.
Change Origin	N/A.	*ChangeSourceIntention, ChangeTargetIntention*	N/A.
Retype	*RetypeIntention*	*RetypeStrategy*	N/A.
Give	*GiveListofParameters*	N/A	*GivePre/ PostCondition, GiveBusinessRule*
Withdraw	*WithdrawListofParameters*	N/A.	*WithdrawPre/ PostCondition, WithdrawBusinessRule*
Modify	*Modify (Verb, Target, List of Parameters)*	N/A.	*ModifyPre/PostCondition, ModifyBusinessRule*

A more precise definition of each of the 32 operators shown in Table 2 is provided in the Appendix. The definition of each of the operators is composed of a *signature* and a *predicate*. The signature identifies the type of the elements involved in the As-Is map (before the operator is executed), and in the To-Be map (after the execution of the operator). The predicate is a first-order logic formula specifying the conditions that must be fulfilled in the To-Be map. The predicate ensures the *correctness* property of the map gap typology: it guarantees that the operator leaves the To-Be map in a correct state, i.e. a state which preserves the map invariants. For example, the *AddIntention* is defined as follows:

Signature $\text{AddIntention} : \text{Intention}^2 \rightarrow \text{Strategy}^2,\ \text{Intention}$

Predicate $\text{AddIntention}(I_1, I_2) = \text{St}_1.\text{has}-\text{for}-\text{source}(I_1)\wedge \text{St}_1.\text{has}-\text{for}-\text{target}(I)\wedge \text{St}_2.\text{has}-\text{for}-\text{source}(I)\wedge \text{St}_2.\text{has}-\text{for}-\text{target}(I_2)|(\text{St}_1, \text{St}_2) \in \text{Strategy}, I \in \text{Intention}$

The addition of the new intention I in the To-Be map ensures that there exists at least one section ($<I_1$,I,St_1 $>$) in which I is the target intention and one section ($<$I, I_2, St_2 $>$) in which I is the source intention. Therefore, the predicate preserves the invariants I2 & I3.

4 The gap elicitation process

The gap elicitation process draws from the top-down requirements elicitation process in which high-level, strategic goals are reduced to low-level, operationalisable goals. Similarly the gap elicitation process starts with the elicitation of gaps between the top-level As-Is and To-Be maps. The refinement mechanism of map sections into maps (Sect. 2) is used as a means to study gaps at different levels of detail. The refinement of sections of the To-Be map allows us to reduce single gaps expressed between top-level maps into a set of gaps between the refined As-Is and To-Be sections. The process continues until refined maps contain operationalisable intentions and strategies. These map gaps can then be expressed as schema gaps (see Fig. 1). It is therefore through the refinement process that the gap granularity issue is handled.

More precisely, the process for eliciting gaps is an iterative one as follows:

– Repeat until all maps have been considered.
1. Construct the As-Is map.
2. Construct the To-Be map & Identify gaps between maps.
3. Deliberate &Commit.

The three steps are carried out in a *participative* manner. This allows the consideration of different viewpoints [29, 30] with the aim of reconciling them co-operatively, in the construction of the As-Is and To-Be maps as well as in the elicitation of gaps. Additionally, in step 3, the decision to refine elicited gaps in an iteration is also made co-operatively. As before, the refinements committed to in this step emerge as a consensus from among the different viewpoints.

Each iteration is related to *one single To-Be map* and includes three key activities to:

1. Construct the *As-Is map* if it does not exists yet.
2. Construct the To-Be by difference with the As-Is map, taking into account the target selected system and the organisation requirements for change. The *To-Be map* and the *Gaps* are modelled concurrently and then, documented.
3. *Deliberate on each section* of the To-Be map to decide if further refinement is required to identify more detailed gaps or not. Every section marked as '*to-be-refined*' will serve as starting point for a new iteration

of the elicitation process. Every section that does not require refinement gets the 'green' status.

The suggested process is clearly *top-down* and *participative*:

- The top-down nature coupled with the refinement mechanism permits the handling of the subdivision of a single gap into other gaps.
- The participative nature facilitates a discussion of gap elicitation and decision-making on gap refinement.

5 Illustrating the process with the DIAC case study

This section illustrates the process for eliciting gaps by reporting activities run with the DIAC company. To set the case study into context, the top-level map describing DIAC's main requirements to be supported by the future software system is first described. One of the sections of the map is then refined to illustrate the three activities of one iteration of the process.

5.1 DIAC top-level map

The overall objective of the Renault-DIAC company is to sell financing products associated to vehicles manufactured by the Renault group. These products are credits taking the form of loans for purchasing vehicles (with or without purchase option), and leases. DIAC's business processes are traditionally centred on sales and post-sales administration of contracts made with customers. Sales processes involve building catalogues of products and making contracts with customers. The post-sales processes include treasury and information flow management.

The requirements for the future software support of DIAC's business processes are identified in the *Finance the purchase and lease of Renault Vehicles* To-Be map shown in Fig. 4. Three intentions are emphasised to define the future situation: *Offer a product*, *Gain a customer,* and *Manage the customer relationship*.

To *Gain a customer* is done *By prescription* of the products offered by the company, *By prospecting* new customers, and *By securing the customer loyalty*. This strategy is particularly important as it supports the company's essential requirement to keep customers as long as there is no need to *Stop* financing them *by exclusion*.

Manage the customer relationship is initiated *By demand of transfer* of the contracts signed with the pre-sales department to the post-sales administration. In DIAC's vision of the future way to hold the business, 'customer relationship' means having business dealings with, and for customers. The intention name was thus introduced to emphasise a determining gap with the contract-wise management of customers currently achieved in Spain.

Managing the relationship with customers should be done *By debts recovery* according to the contracts repayment schedules, and by managing multiple flows of customer-related information. This is shown in the map by the strategies: *By processing modification requests*, *By processing information and complaints requests*, and *By handling legal obligations of communication*. The latter strategy is imposed by the European and national laws on information privacy. Managing the customer relationship *By capitalisation* of treasury is an absolute requirement to ensure forthcoming financing. The strategy *By handling accidents* is important as well, as for some products DIAC may propose to pay in the place of customers who have suffered damages that stop them reimbursing their debts.

Let us assume that at the end of the iteration related to the To-Be map shown in Fig. 5, the section $<$*Offer a product, Gain the customer, By prescription*$>$ was left with a 'to-be-refined' status and let us illustrate the three activities occurring in one single iteration by reporting respectively: (1) the refinement of the section C3 by the construction of the As-Is map *Gain the customer by prescription*, (2) the identification of gaps between the current and future situations by the construction of a To-Be map, and (3) commenting on the decisions made about how to proceed with the gaps identified.

5.2 Constructing the As-Is map to *gain the customer by prescription*

The construction of an As-Is map is not as straightforward as we would like it to be. We developed different strategies supported by guidelines to help in the construction of a map and improve its quality. As an illustration of this methodological aspect we present in Figs 6 and 7 respectively the initial map for *Gain the customer by prescription* and that resulting from the application of quality guidelines. The strategy used in DIAC was a participative strategy: domain experts were providing the domain knowledge that we were modelling in maps.

Today, when a Spanish customer goes to a DIAC concessionaire to be prescribed a financing product, he/she deals with salesmen whose first concern is to gather data for the offer they will use to *Make a contract with the customer*. To *Gather the offer data*, Spanish salesmen get assistance from the software system to (a) select a product from DIAC's catalogue, (b) formulate an offer based on this product, and (c) complete the offer with customer personal information and other data (the repayment schedule details, the price and nature of the financed vehicle).

Before the contract is made, pre-sales administrators must intervene in the process to *Evaluate the risk* represented by each offer. This helps ensure that a maximum number of credits will be correctly paid back. Then, they *Decide on the offer*. This results in the validation of the offer, in its rejection, or in a counter-offer aimed at maximising the number of customers with which salesmen can make a contract.

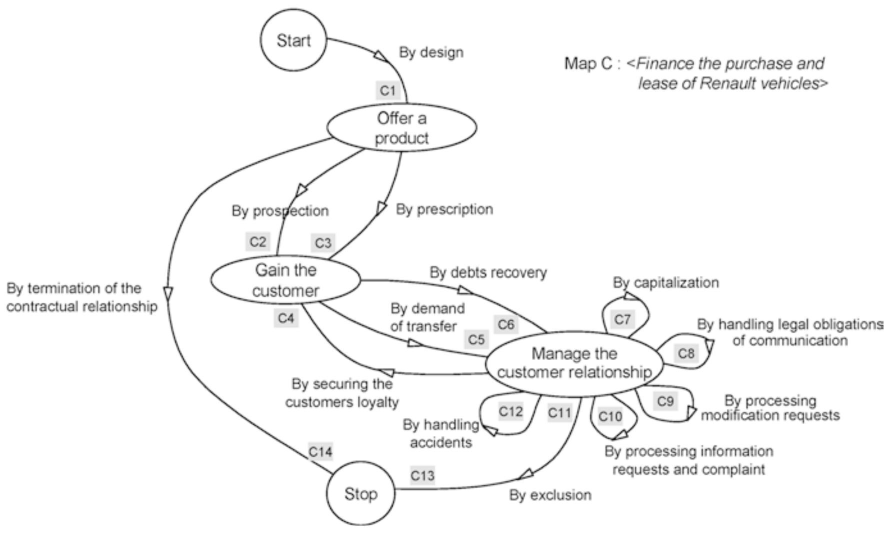
Start
By design
C1
Map C : <Finance the purchase and lease of Renault vehicles>
Offer a product
By prospection
By prescription
C2
C3
By termination of the contractual relationship
Gain the customer
By debts recovery
By capitalization
C4
By demand of transfer
C6
C7
C5
By handling legal obligations of communication
C8
Manage the customer relationship
By securing the customers loyalty
C12
C11
C9
By processing modification requests
By handling accidents
C10
By processing information requests and complaint
C14
C13
Stop
By exclusion

Fig. 5 Top level To-Be map of DIAC

These intentions are supported in the current situation by several features of the Spanish software system. These are shown in Fig. 6 through different strategies. For example, an expert system generating evaluations of the customers' reliability helps the pre-sales administrators to *Evaluate the risk Automatically*. The heuristic rules implemented in this software component use a national banking proscription list and, in particular, scoring functions based on the offer data. The source of the corresponding strategy is thus the intention *Gather offer data*.

Spanish salesmen and pre-sales administrators are not the only users to be involved in the process. Independent agents selling second-hand vehicles and Spanish employees of DIAC may also gather offer data. However, the software proposes them reduced features (e.g. agents can only offer second-hand vehicle loan products) through specific media (e.g. VT100 terminals and Intranet); hence the two additional strategies to gather offer data.

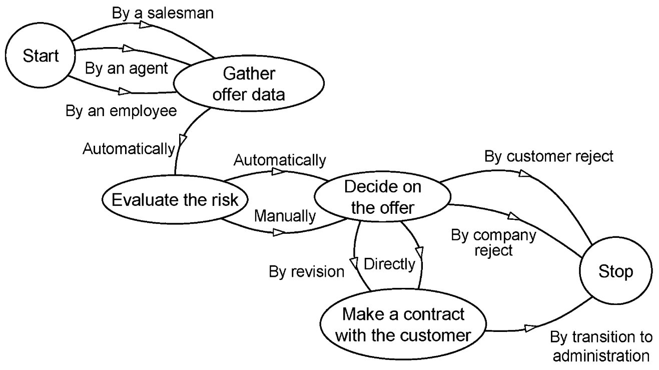

Fig. 6 Initial As-Is map to *Gain the customer by prescription*

European legislation imposes that offers are not rejected in a fully automated way. Deciding on the offer *By manual risk evaluation* as well as *Automatically* is necessary both to complete the evaluation performed by the expert system and to revise offers based on modified data. Once the decision is made, the customer can reject the offer. Besides, the company can at any time decide to definitively end the process if it appears that the customer is a crook.

To make a contract with the customer, a vendor (i.e. salesman or an agent) *Directly* prints the contract and records its signature in the software. The process stops *By transition to administration* of the contract. This can occur several times if the contract is made again *By revision* of minor data such as the vehicle registration number.

The initial map shown in Fig. 6 was improved by applying *quality rules* which are associated to the map formalism [19]. The resulting map in Fig. 7 shows a number of adaptations that followed from the application of these rules. For example, rule R4 (sections representing mutually exclusive ways to produce the same result shall be bundled) was applied and resulted in the bundling of the sections < *Start, Gather the offer data, By an agent* >,

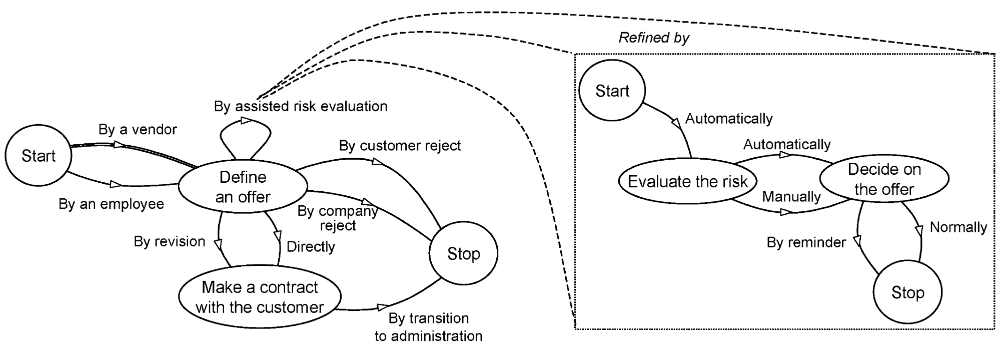

Fig. 7 Improved As-Is map to *Gain the customer by prescription* (left), refined by the map *Define an offer by assisted risk evaluation* (right)

and < *Start, Gather the offer data, By a salesman* > into < *Start, Gather the offer data, By a vendor* > .

Based on the business fact that the intentions *Gather the offer data* and *Decide on the offer* complement each other, the two intentions were aggregated as rule R6 suggests (intentions that mutually complement each another shall be aggregated in one intention which abstracts them). This resulted in the intention *Define an offer* which better abstracts the actual purpose of the company's vendors.

Besides, the experts proposed to consider the intentions *Evaluate the risk* and *Decide on the offer* as a subset of *Define an offer*. Rule R1 (no intention in a map shall be the subset of another one) was thus applied, which made the former intentions disappear from the map.

However, in order to differentiate the vendor's participation in defining an offer from the pre-sales administrator's part, the resulting map was adapted by adding the strategy *By assisted risk evaluation*. As Fig. 7 shows, the resulting section was refined by a map containing the subset intentions and strategies of the section < *Define an Offer, Define an Offer, By assisted risk evaluation* > .

Once the As-Is map constructed, the next activity is aimed to construct the To-Be map and to identify gaps.

5.3 Constructing the To-Be map through gap identification

The To-Be map presented in Fig. 8 describes the requirements for the software system supporting the goal to *Gain the customer by prescription* in the future situation. Although it contains similarities with the As-Is map, this map is different from the map of Fig. 7. The gaps between the As-Is map and the To-Be map are identified in Table 3. For each intention, strategy and section involved in a gap, the operator used to define the To-Be map is quoted in a separate line.

Table 3 shows, for example, that the intention *Define an offer* was replaced by *Prepare a contract*. Indeed, in the current situation, vendors *Define an offer* based on a catalogue of products describing services, pricing rules and the terms to use in contracts. To this product-oriented view vendors opposed a customer-oriented view in which the software would offer them the ability to *Collect a request* from a customer rather than to *Define an offer*. By adding this intention, vendors expressed their requirement of a system support to discuss requests with customers, envisage different products according to the customers' needs (new products such as reserve of money but also other services such as vehicle maintenance, petrol cards, insurance, etc.) and propose complementary products in packages.

The company's Web site already offers a simulation feature based on the up-to-date catalogue of offers. However, its success suggests more advanced features to be provided. The future system component to *Collect a request* is intended to be used *By a vendor*, but can also bring competitive advantage to the company if directly used *By customers via Internet*. The vendors asked to offer direct access to customers. The features offered to customers should, however, be designed so that they keep contacting vendors. The origin of the former strategy was thus changed and the latter strategy added.

The system feature supporting the *Collect a request By a customer via Internet* is duplicated with the Intranet facility provided to the Spanish employees. In order to avoid the maintenance costs of the Intranet facility, the corresponding strategy *By an employee* was removed from the map. In the future situation, employees will use the same facility as any other customer (still with special price grids though).

Whereas in the Spanish branch of DIAC the business and current software differentiate offers and contracts, French vendors consider that offers are already contracts that are not yet signed. The experts agreed that this terminology difference would have an important impact at the operational level because the objects and business rules involved would be very different. In order to homogenise the business terminology across the countries in which the DIAC company is settled, it was decided to replace the intention *Define an offer* by *Prepare a contract*. The following shows that this gap had a multiple impact on the other parts of the To-Be map under construction.

Contracts must still be initially prepared based on a risk evaluation. It was thus necessary to add the section

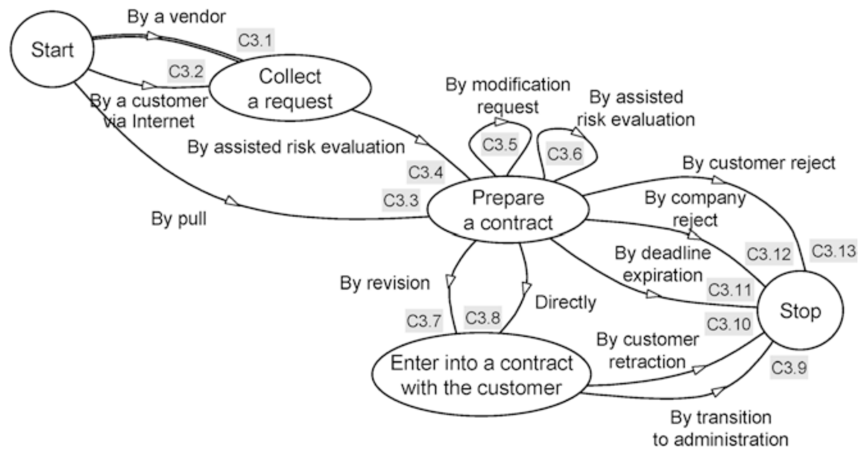

Fig. 8 To-Be map describing the requirements for the future software system support to *Gain the customer by prescription*

Table 3 Gaps between the As-Is and To-Be maps to *Gain the customer by prescription*

Operator	Intention	Strategy	Section
Rename	*Make a contract with the customer* as *Enter into a contract with the customer*		
Add	*Collect a request*	*By a customer via Internet* *By pull* *By modification request* *By customer retraction*	*< Collect a request, Prepare a contract, By assisted risk evaluation >*
Remove		*By an Employee*	
Split		*By company reject* into *By company reject* and *By deadline expiration*	
Change Origin		*By a vendor* target intention to *Collect a request*	
Replace	*Define an offer* by *Prepare a contract*		

< Collect a request, Prepare a contract, By assisted risk evaluation >.

There was a strong requirement to support customer requests for contract modification in the future. Contracts should be recorded in the software system with a 'recorded' or 'scored' state and with a recommendation such as 'to accept', 'to accept with conditions', 'to reject', or 'to study'. The state might be revised each time a modification is requested. The strategy *By modification request* was thus added with the consequence of keeping the loop section *< Prepare a contract, Prepare a contract, By assisted risk evaluation >*.

The opportunity to prepare predefined contracts based on profiling facilities in order to attract customers was then identified. This opportunity was seized by making the decision to add the strategy *By pull* in the To-Be map.

Vendors asked that the future software system automatically puts unsigned contracts in a 'without response' state after deadlines defined in contracts, or at the end of a product life cycle (e.g. for event-related products such as Christmas packages). The issues raised by this specific way to stop a contract had to be explored separately from the *By the company reject* strategy which, in both the current and future situation, corresponds to the decision of excluding customers suspected to be crooks. It was decided to split the strategy *By company reject,* hence the *By deadline expiration* strategy in the To-Be map. While it was agreed to reuse the former strategy, the latter had to be detailed further.

In order to avoid confusion between the intention *Make a contract with the customer* and the fact that in the future situation contracts should actually be created with the software while *Prepar(ing) a contract*, it was decided to rename the former intention as *Enter into a contract with the customer*. The experts agreed that this should have no further impact.

The last gap identified in Table 3, namely adding the *By customer retraction* strategy to stop the process, implements the French legislation according to which any customer should be able to cancel a contract for a certain period following its signature. Then, if sub-contractors and vendor commissions have already been paid, a reimbursement should be asked for.

5.4 Deliberating and committing

Once the gaps have been identified and documented, the sections of the To-Be maps were scrutinised to determine if a further analysis of requirements and an in-depth detection of gaps was necessary. The decision was finally to perform the refinement of four sections, namely C3.2, C3.3, C3.4 and C3.6. These sections were marked with the 'to-be-refined' status whereas all the others were marked with the 'green' status. The former will be subject to a similar study to the one we reported on in the last three sub-sections whereas the gaps, if any, in the latter will be propagated at the software schema level (see Fig. 1).

6 Conclusion

System adaptation is done under intense time pressure: the new system must be put in place yesterday! Therefore, it is not possible to develop a To-Be model *from scratch*, given the time and resources involved. A workable strategy under these circumstances is to use and modify what is available, and add the remaining. This is the thrust of the gap drive proposed in this paper. In the DIAC project, our 'guesstimate' is that we gained about 50% in time over the traditional roach. The gap drive seems to be beneficial both from the time and economics points of view.

When requirements change, then the stakeholders need to negotiate these changes with each other. In conformity with traditional wisdom, an explicit statement of these facilitates negotiation. It clarifies what has to change and why, and acts as an agreed contract on the changes. It is also a specification of software changes to be carried out. As shown in the DIAC example presented in the paper, map gaps provide stakeholders the negotiation platform. Additionally, map gaps constitute a specification of the software system changes: DIAC change requirements were, in fact, used to control the outsourcing of the software adaptation work.

Gap elicitation is an intellectually demanding process. We believe that the choice of the map formalism contributes rather significantly to managing this process. Being essentially intentional or goal-oriented, the map provides a high-level expression of gaps. This reduces the plethora of changes to a manageable number and reduces the complexity of the gap elicitation process.

We are conscious of the lack of consideration given to the identification of conflict situations that could arise during the gap elicitation process. This is one of the further steps of our research agenda.

Acknowledgements Sincere thanks are due to the anonymous reviewers who provided constructive criticism that has made improvements possible.

7 Appendix: Defining map change operators to specifying map gaps

7.1 Rename

7.1.1 RenameIntention

As any *element* [Fig. 2] an intention is associated to a *string* that defines its *name*. This holds both before and as result of applying the *RenameIntention* operator to a given intention.

$$\text{RenameIntention} : \text{Intention} \rightarrow \text{String}$$
$$\text{RenameIntention(I)} = \text{I.name(N)}|\text{N} \in \text{String}$$

7.1.2 RenameStrategy

As any *element*, a strategy has a *name* which is a *string*. Therefore, renaming a strategy is achieved by changing the string value that defines its *name*.

$$\text{RenameStrategy} : \text{Strategy} \rightarrow \text{String}$$
$$\text{RenameStrategy(St)} = \text{St.name(N)}|\text{N} \rightarrow \text{String}$$

7.1.3 RenameSection

Names are given to sections as a means to refer to them more easily. Renaming a section is, therefore, not a very significant gap defined as follows:

```
RenameSection: Section → String
RenameSection(Se) = Se.name(N) | N ∈ String
```

7.2 Add

7.2.1 AddIntention

As a result of map *invariants* (section 3), any map intention is connected to other intentions through strategies. Therefore, when an intention I is added, it must be related by strategies to at least two other intentions of the map (I_1 & I_2 in the predicate). In order to preserve C2, a strategy St_1 must have the added intention I as target intention and an I_1 intention as source intention. To preserve C3, another strategy St_2 must have the added intention I as its source and I_2 as its target. By default I_1 and I_2 might be Start and Stop, respectively.

$$\text{AddIntention} : \text{Intention}^2 \rightarrow \text{Strategy}^2, \text{Intention}$$
$$\text{AddIntention}(I_1, I_2) = St_1.\text{has}-\text{for}-\text{source}(I_1)\wedge$$
$$St_1.\text{has}-\text{for}-\text{target}(I)\wedge$$
$$St_2.\text{has}-\text{for}-\text{source}(I) \wedge St_2.\text{has}-\text{for}-\text{target}(I_2)|$$
$$\times\,(St_1,\ St_2) \in \text{Strategy},$$
$$I \in \text{Intention}$$

7.2.2 AddStrategy

According to the map meta-model, any strategy in a map can only be defined between two intentions: its *source intention* and its *target intention* respectively. Therefore, when a strategy is added it must be linked to two existing intentions, one being the *source intention* and the other being its *target intention*.

$$\text{AddStrategy} : \text{Intention}^2 \rightarrow \text{Strategy}, \text{Section}$$
$$\text{AddStrategy}(I_1, I_2) = \text{St.has}-\text{for}-\text{source}(I_1)\wedge$$
$$\text{St.has}-\text{for}-\text{target}(I_2)|\text{St} \in \text{Strategy}$$

7.2.3 AddSection

In order to preserve the invariant I3 and to conform to the corollary C5 the *AddSection* operator ensures that the added section (I_s, I_t, St) is connected to the rest of the map. This is achieved by enforcing the existence of (a) the strategy St_1 having I_s as its target and an intention I_1 already existing as its source and (b) the strategy St_2 having I_t as its source and an intention I_2 already existing as its target. By default, I_1 and I_2 may be the *Start* and *Stop* intentions, respectively.

$$\text{AddSection} : \text{Intention}^2 \rightarrow \text{Section}, \text{Strategy}^3, \text{Intention}^2$$
$$\text{AddSection}(I_1, I_2) = St_1.\text{has}-\text{for}-\text{source}(I_1)\wedge$$
$$St_1.\text{has}-\text{for}-\text{target}(I_s)\wedge$$
$$St_2.\text{has}-\text{for}-\text{source}(I_t) \wedge St_2.\text{has}-\text{for}-\text{target}(I_2)|$$
$$\times\,\text{Se.composed\_of(St)}\wedge$$
$$\text{Se.composed\_of}(I_s) \wedge \text{Se.composed\_of}(I_t), (\text{St}, St_1, St_2)$$
$$\in \text{Strategy},$$
$$(I_s, I_t, I_1, I_2) \in \text{Intention}, \text{Se} \in \text{Section}.$$

7.3 Remove

7.3.1 RemoveIntention

The execution of the *RemoveIntention* is concerned with the corollaries of map *invariants* C1, C2 and C3. Preserving them implies that any strategy St_i^s(or) St_j^t having the removed intention I as a source intention (or target intention) must henceforth have another source intention By default the *Start* (or *Stop*) intentions are used.

$$\text{RemoveIntention} : \text{Intention}, \{\text{Strategy}\}, \{\text{Strategy}\}, \{\text{Intention}\}, \{\text{Intention}\} \to \varnothing$$

$$\text{RemoveIntention}(\text{I}, \{\text{St}_i^s\}, \{\text{St}_j^t\}, \{\text{I}_k^s\}, \{\text{I}_l^t\}) = [\forall i, \text{St}_i^s.\text{has}-\text{for}-\text{source}(\text{I}_k^s) | \text{I}_k^s \in \{\text{I}_k^s\}] \wedge [\forall j, \text{St}_j^t.\text{has}-\text{for}-\text{target}(\text{I}_l^t) | \text{I}_l^t \in \{\text{Itl}\}].$$

7.3.2 *RemoveStrategy*

Let assume that the section (I_s, St, I_t) exists in the As-Is map. In order to preserve C1, C2 or C3 the *Remove Strategy* operator applied to the strategy St imposes the existence of two sections (Is, St_1, I_1) and (I_2,St_t, I_t). I_s and I_t remain therefore source intention (or target intention) of sections in the map. By default I_1 and I_2 can be *Start* and *Stop* respectively.

$$\text{RemoveStrategy} : \text{Strategy}, \text{Intention}^4 \to \text{Strategy}^2$$

$$\text{RemoveStrategy}(\text{St}, \text{I}_s, \text{I}_t, \text{I}_1, \text{I}_2) = \text{St}_1.\text{has}-\text{for}-\text{source}(\text{I}_s) \wedge \text{St}_1.\text{has}-\text{for}-\text{target}(\text{I}_1) \wedge \text{St}_2.\text{has}-\text{for}-\text{source}(\text{I}_2) \wedge \text{St}_2.\text{has}-\text{for}-\text{target}(\text{I}_t) | \times (\text{St}_1, \text{St}_2) \in \text{Strategy}$$

7.3.3 *RemoveSection*

When a section Se (<I_s,I_t,St>) is removed, its three components (the intentions I_s & I_t and the strategy St) are removed. In order to preserve the map invariants, the *Remove Section* operation ensures that any strategy St_i^s having I_s or I_t as source intention has in the To-Be map another intention I_k^s (*Start* by default) as source. The same holds for strategies St_j^t having I_s or I_t as target intention. After the section removal, these strategies must have another target intention I_l^t (*Stop* by default).

$$\text{RemoveSection} : \text{Section}, \{\text{Strategy}\}^2, \{\text{Intention}\}^2 \to \varnothing$$

$$\text{RemoveSection}(\text{Se}, \{\text{St}_i^s\}, \{\text{St}_j^t\}) = [\forall i, \text{St}_i^s.\text{has}-\text{for}-\text{source}(\text{I}_k^k) | \text{I}_k^s \wedge \{\text{I}_k^s\}] \wedge [\forall j, \text{St}_j^t.\text{has}-\text{for}-\text{target}(\text{I}_l^t) | \text{I}_l^t \in \{\text{I}_l^t\}].$$

7.4 Merge

7.4.1 *MergeStrategy*

A strategy St can be considered as the result of merging two strategies St_1 and St_2 iff these have the same source intention I_s and the same target intention I_t. If this condition is not met, then a different operator must be involved in the gap elicitation. For instance, if three different intentions are involved, the *MergeSection* operator can be used to elicit the gap.

$$\text{MergeStrategy} : \text{Strategy}^2, \text{Intention}^2 \to \text{Strategy}$$

$$\text{MergeStrategy}(\text{St}_1, \text{St}_2, \text{I}_s, \text{I}_t) = \text{St}.\text{has}-\text{for}-\text{source}(\text{I}_s) \wedge \text{St}.\text{has}-\text{for}-\text{target}(\text{I}_t) | \text{St} \in \text{Strategy}$$

7.4.2 *MergeIntention*

When two intentions I_1 and I_2 are merged, the intention I replaces I_1 and I_2 in any section having initially I_1 or I_2 as source intention or target intention.

$$\text{MergeIntention} : \text{Intention}^2, \{\text{Strategy}\}^2 \to \text{Intention}$$

$$\text{MergeIntention}(\text{I}_1, \text{I}_2, \{\text{St}_i^s\}, \{\text{St}_j^t\}) = [\forall i, \text{St}_i^s.\text{has}-\text{for}-\text{source}(\text{I}_r)] \wedge [\forall j, \text{St}_j^t.\text{has}-\text{for}-\text{target}(\text{I}_r)] | \text{I}_r \in \text{Intention}.$$

7.4.3 *MergeSection*

The merging of two sections Se_1 (<Is_1, It_1, St_1>) and Se_2 (<Is_2, It_2, St_2>) results in a section Se_r (<Is_r, St_r, It_r>). There are pre-conditions for applying this operator: Is_1 must be different from Is_2 or It_1 must be different from It_2. In the resulting situation, Is_1 and Is_2 are replaced by Is_r and It_1 and It_2 are replaced by It_r in any section having initially Is_1, Is_2, It_1 or It_2 as source or target intention.

$$\text{MergeSection} : \text{Section}^2, \{\text{Strategy}\}^2 \to \text{Section}, \text{Intention}^2, \text{Strategy}$$

$$\text{MergeSection}(\text{Se}_1, \text{Se}_2, \{\text{St}_i^s\}, \{\text{St}_j^t\}) = \{\forall i, \text{St}_i^s.\text{has}-\text{for}-\text{source}(\text{I}_r)\} \wedge \{\forall j, \text{St}_j^t.\text{has}-\text{for}-\text{target}(\text{I}_r)\} \wedge \text{Se}_r.\text{composed\_of}(\text{Is}_r) \wedge \text{Ser}.\text{composed\_of}(\text{It}_r) \wedge \text{Ser}.\text{composed\_of}(\text{St}_r) | \times \text{St}_r \in \text{Strategy}, (\text{Is}_r, \text{It}_r) \in \text{Intention}, (\text{Se}_r) \in \text{Section}$$

7.5 Split

7.5.1 *SplitStrategy*

Splitting a strategy St of a section <I_s, I_t, St> results in two strategies St_1 and St_2. Defining the sections <I_s, I_t, St_1> and <I_s, I_t, St_2>.

$$\text{SplitStrategy} : \text{Strategy}, \text{Intention}^2 \to \text{Strategy}^2$$

$$\text{SplitStrategy}(\text{St}, \text{I}_s, \text{I}_t) = \text{St}_1.\text{has}-\text{for}-\text{source}(\text{I}_s) \wedge \text{St}_2.\text{has}-\text{for}-\text{source}(\text{I}_s) \wedge \text{St}_1.\text{has}-\text{for}-\text{target}(\text{I}_t) \wedge \text{St}_2.\text{has}-\text{for}-\text{target}(\text{I}_t) | \times (\text{St}_1, \text{St}_2) \in \text{Strategy}$$

7.5.2 *SplitIntention*

The *SplitIntention* operator permits the replacement of one intention I by two intentions I_1 and I_2. In order to keep the invariants satisfied, the operator ensures that any strategy St_j^t having the intention I for target in the As-Is map has either I_1or I_2 as target in the To-Be map. The same holds for strategies St_i^s that initially had I as source intention.

$$\mathrm{SplitIntention} : \mathrm{Intention}, \{\mathrm{Strategy}\}^2 \rightarrow \mathrm{Intention}^2$$

$$\mathrm{SplitIntention}(I, \{St_i^s\}, \{St_j^t\})$$
$$= [\forall i, St_i^s.has - for - source(I_1) \wedge$$
$$St_i^s.has - for - source(I_2)] \wedge [\forall j, St_j^t.has - for - target(I_1) \vee$$
$$St_j.has - for - target(I_2)]|(I_1, I_2) \in \mathrm{Intention}$$

7.5.3 *SplitSection*

A section Se (< Is, It, St >) can be split into two sections Se_1 ($<Is_1, It_1, St_1>$) and Se_2 ($<Is_2, It_2, St_2>$). This implies as shown by the predicate that

- any strategy St_i^s having initially I_s (or I_t) as source must have Is_1 or Is_2 (or It_1 or It_2) as source in the final situation. Similarly,
- any strategy St_j^t having initially I_s (or I_t) as target must have Is_1 or Is_2 (or It_1 or It_2) as target in the final situation.

$$\mathrm{SplitSection} : \mathrm{Section}, \{\mathrm{Strategy}\}^2$$
$$\rightarrow \mathrm{Section}^2, \mathrm{Intention}^2, \mathrm{Strategy}^2$$

$$\mathrm{SplitIntention}(Se, \{St_i^s\}, \{St_j^t\})$$
$$= [\forall i, St_i^s.has - for - source(Is_1) \vee$$
$$St_i^s.has - for - source(Is_2) \vee St_i^s.has - for - source(It_1)$$
$$\vee St_i^s.has - for - source(It_2)] \wedge$$
$$[\forall j, St_j^t.has - for - target(Is_1)$$
$$\vee St_j^t.has - for - target(Is_2)$$
$$\vee St_j^t.has - for - target(It_1) \vee$$
$$St_j^t.has - for - target(It_2)] \wedge Se_1.composed\_of(Is_1)$$
$$\wedge Se_1.composed\_of(It_1) \wedge$$
$$Se_1.composed\_of(St_1) \wedge Se_2.composed\_of(Is_2)$$
$$\wedge Se_r.composed\_of(It_2) \wedge$$
$$Se_2.composed\_of(St_2)|(Is_1, Is_2, It_1, It_2)$$
$$\in \mathrm{Intention}, (St_1, St_2) \in \mathrm{Strategy},$$
$$(Se_1, Se_2) \in \mathrm{Section}$$

7.6 ChangeOrigin

The *ChangeOrigin* operator execution permits a source intention (or a target intention) I of a strategy St to be replaced by another intention I_n. When the origin of a strategy is changed, the invariant I3 and the corollaries C1, C2 and C3 may not hold in the To-Be map. This issue is solved by ensuring that there exits in this map, a strategy St_n having I as source (or target depending on the initial link).

$$\mathrm{ChangeOrigin} : \mathrm{Strategy}, \mathrm{Intention}^2 \rightarrow \mathrm{Strategy}$$

$$\mathrm{ChangeOrigin}(St, I, I_n) = [St.has - for - source(I_n)$$
$$\vee St.has - for - target(I_n)] \wedge$$
$$[St_n.has - for - source(I) \vee St_n.has - for - target(I)|$$
$$\times St_n \in \mathrm{Strategy}$$

7.7 Retype

7.7.1 *RetypeStrategy*

The gap between a As-Is map and a To-Be map may correspond to the retyping of a As-Is strategy St into a To-Be intention I. In order to satisfy the corollaries C2 and C3, the intention which was source (or target) of St must remain the source intention (or target intention) of at least one strategy in the To-Be map. Furthermore, to preserve C1, the new intention I must not be isolated. A strategy St_3 having I as source and another strategy St_4 having I as target must exist in the To-Be map

$$\mathrm{RetypeStrategy} : \mathrm{Strategy}, \mathrm{Intention}^6 \rightarrow \mathrm{Strategy}^4$$

$$\mathrm{RetypeStrategy}(St, I_s, I_t, I) = St_1.has - for - source(I_s)$$
$$\wedge St_1.has - for - target(I_1) \wedge$$
$$St_2.has - for - target(I_t) \wedge St_2.has - for - source(I_2)$$
$$\wedge St_3.has - for - source(I) \wedge$$
$$St_3.has - for - target(I_3) \wedge St_4.has - for - target(I) \wedge$$
$$St_4.has - for - target(I_4)|(I_1, I_2, I_3, I_4)$$
$$\in \mathrm{Intention}(St_1, St_2, St_3, St_4) \in \mathrm{Strategy}$$

7.7.2 *RetypeIntention*

Vice-versa, an As-Is intention I can be retyped into a To-Be strategy St. The predicate of the *RetypeIntention* operator ensures that any strategy St_i^s having initially I as source (or target) must have another source I_k^s (or target I_l^t) intention. The new strategy St must have a source intention I_1 and a target intention I_2 as well in the To-Be map.

$$\mathrm{RetypeIntention} : \mathrm{Intention}^3, \{\mathrm{Strategy}\}, \{\mathrm{Strategy}\},$$
$$\{\mathrm{Intention}\},$$
$$\{\mathrm{Intention}\} \rightarrow \mathrm{Strategy}$$

$$\mathrm{RetypeIntention}(I, I_1, I_2, \{St_i^s\}, \{St_j^t\}, \{I_k^s\}, \{I_l^t\}) =$$
$$[\forall St_i \in \{St_i^s\}, St_i.has - for - source(I_k) \wedge I_k \in \{I_k^s\}] \wedge$$
$$[\forall St_j \in \{St_j^t\}, St_j.has - for - target(I_l) \wedge I_l \in \{I_l^t\}]$$
$$\wedge St.has - for - source(I_1) \wedge$$
$$St.has - for - target(I_2)|St \in \mathrm{Strategy}.$$

The *Give*, *Withdraw* and *Modify* operators apply on properties of *Intentions* (*Verb, Target, Parameters*) and properties of *Sections* (*Business rules, Pre-condition, Post-condition*). Any change of properties maintains the map *invariants*. Therefore, there is no need for a predicative definition of these operators.

References

1. Lientz BP, Swanson, EB (1980) Software maintenance management, a study of computer application software in 487 data processing organizations. Addison-Wesley, Reading, MA
2. Nosek JT, Palvia P (1990) Software maintenance management: changes in the last decade. J Softw Maint 2(3), 157–174
3. IPSE International Workshop on the Principles of Software Evolution
4. Breche P (1996) Advanced primitives for changing schemas of object databases. In: Proceedings of the international conference CAiSE'96, Heraklion, Greece. Springer, Berlin Heidelberg New York
5. Lauteman SE (1997) Schema versions in object-oriented database systems. In: Proceedings of the fifth international conference on database systems for advanced applications, Melbourne, Australia, 1–4 April 1997
6. Banerjee J, Kim W, Kim H-J, Korth HF (1987) Semantics and implementation of schema evolution in object oriented databases In: Proceedings of the ACM-SIGMOD annual conference, San Francisco, CA, May 1987, pp 311–322
7. Casati F, Ceri S, Pernici B, Pozzi G (1996) Workflow evolution. In: Proceedings of 15th international conference on conceptual modeling (ER'96), Cottbus, Germany, pp 438–455
8. Liu C, Orlowska M, Li H (1998) Automating handover in dynamic workflow environments. In: 10th conference on advanced information systems engineering, Pisa, Italy
9. Sadiq S (2000) Handling dynamic schema change in process models. In: Australian Database Conference, Canberra, Australia
10. Bandinelli S, Fuggetta A, Ghezzi C (1993) Software process model evolution in the SPADE environment. IEEE Trans Softw Eng 19(12), 1128–1144
11. Heimann P, Joeris G, Krapp C, Westfechtel B (1996) DYNAMITE: dynamic task nets for software process management. In: Proceedings of the 18th international conference on software engineering (ICSE 18), Berlin, Germany, March 1996, pp 331–341
12. Si-Said S, Rolland C, Grosz G (1996) MENTOR: a computer aided requirements engineering environment. In: Proceedings of the international conference CAiSE'96. Heraklion, Greece. Springer, Berlin Heidelberg, New York
13. Lehman MM, Ramil JF, Kahen G (2001) Thoughts on the role of formalisms in studying software evolution. In: Mens T, Wermelinger M (eds) International special session on formal foundations of software evolution, Lisbon, Portugal, March 2001. Co-located with the European conference on software maintenance and reengineering (CSMR 2001)
14. Jarke M, Pohl K (1993) Establishing visions in context: toward a model of requirements processes. In: Proceedings of the 12th international conference information systems, Orlando, FL
15. Jackson M (1995) Software requirements and specifications. Addison-Wesley, Reading, MA
16. von Lansweerde A (2000) Requirements engineering in the year 2000: a research perspective. In: 22nd international conference on software engineering
17. Dardenne A, von Lansweerde A, Fickas S (1993) Goal directed requirements acquisition. Sci Comput Program 20, 3–50
18. Jarke M, Pohl K (1994) Requirements engineering in 2001: managing a changing reality. IEEE Softw Eng J November, 257–266
19. Rolland C, Prakash N (2001) Matching ERP system functionality to customer requirements. In: Proceedings of RE'01, 5th international symposium on requirements engineering, Toronto, Canada, pp 66–75
20. Rolland C, Prakash N, Benjamen A (1999) A multi-model view of process modelling. Requirements Eng 4: 169–187
21. Kradolfer M, Geppert A (1999) Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In: Proceedings of the fourth IFCIS international conference on cooperative information systems (CoopIS99). Edinburgh, UK, 2–4 September 1999
22. Information technology–information resource dictionary system (IRDS) (1990) Framework, ISO/IEC International Standard
23. Marttiin P (1994) Methodology engineering in CASE shells: design issue and current practice. PhD thesis, Computer science and information systems reports, Technical report TR-4
24. Grundy JC, Venacle JR (1992) Towards an integrated environment for method engineering. In: Cotterman WW, Senn JA (eds) Challenges and strategies for research in systems development. Wiley, Chichester, pp 45–62
25. Plihon V, Rolland C (1997) Using a generic approach to support the construction of methods. In: Proceedings of the 8th international conference on database and expert systems applications (DEXA'97), Toulouse, France, 1–7 September 1997
26. Prakash N (1999) On method statics and dynamics. Inform Syst 24(8), 613–637
27. Rolland C, Souveyet C., Salinesi C (1998) Guiding goal modelling using scenarios. IEEE Trans Softw Eng (special issue on scenario management) 24(12): 1055–1071
28. Prat N (1997) Goal formalisation and classification for requirements engineering. In: Proceedings of the third international workshop on requirements engineering: foundations of software quality REFSQ'97, Barcelona, pp 145–156
29. Easterbrook S. Resolving requirements conflicts with computer-supported negotiation. In: Jirotka M, Goguen J (eds) Requirement engineering: social and technical issues. Academic Press, New York, pp 41–65
30. Nuseibeh B, Kramer J, Finkelstein A (1994) A framework for expressing the relationships between multiple views in requirements specifications. IEEE Trans Softw Eng 20(10, 760–773