Association for Information Systems
AIS Electronic Library (AISeL)

ACIS 2004 Proceedings | Australasian (ACIS)

December 2004

Towards a Systematic Propagation of Evolution Requirements in IS Adaptation Projects

Camille Salinesi
Université Paris

Anne Etien
Université Paris

Jaana Wayrynen
University of Stockholm/Royal Institute of Technology

Towards a Systematic Propagation of Evolution Requirements in IS Adaptation Projects

Dr Camille Salinesi
Centre de Recherche en Informatique
Université Paris 1 – Panthéon Sorbonne
Email: camille@univ-paris1.fr

Anne Etien
Centre de Recherche en Informatique
Université Paris 1 – Panthéon Sorbonne
Email: aetien@univ-paris1.fr

Jaana Wayrynen
Department of Computer and Systems Sciences
University of Stockholm/Royal Institute of Technology
Email: jaana@dsv.su.se

Abstract

Information system adaptation is a type of system evolution that can be managed defining evolution requirements as a set of gaps with the current system. Today, most Requirements Engineering approaches for system evolution guide the modification of requirements, but very few tell how the required modifications can be elicited or even specified as such in a requirements document. This paper proposes an approach that facilitates the search and expression of evolution requirements. It advocates a business driven approach to align system adaptations to the objectives of the changing organisation. The approach is presented, then and illustrated it with an example.

Keywords

Requirement Engineering, Evolution goal, Business process, Information System, Gaps.

INTRODUCTION

Organisations are more and more confronted with changes in their environment and in their business. To remain competitive they have to adapt, which implies to make their Information Systems (IS) evolve. Indeed, to have an IS that fits the needs of a business, it is necessary that its evolution fits the business evolution too (Salinesi and al. 2003).

IS evolution can be designed as a movement from an initial situation to a future one. These situations are often represented by As-Is models and To-Be models (Jarke and al. 1994).

In many academic approaches, the proposal is to specify requirements about the future situation, then build the corresponding To-Be models. However, our experience showed us that despite the diversity of engineering techniques used in the Industry, people find more efficient in practice to first elicit evolution requirements under the form of *gaps* with the As-Is Business Models (BM) or System Functionality Models (SFM). It is only when people have spotted the gaps with the current situation that they start specifying the To-Be models. The advantage of specifying evolution requirements with gaps is that it allows to avoid re-specifying what remains stable in the system. Besides, it avoids managing multiple versions of requirements documents and complicate traceability when the required evolution is complex. For example, it is possible to specify that a number of required IS changes are required through addition, removal, merging or splitting of a number of elements from the As-Is models rather than to perform all these corresponding modifications in the requirements documents.

We have developed a requirements language based on a typology of gap operators to systematise the formalisation of such evolution requirements. These operators were generalised so that they can be used with any meta-model (Etien and al. 2003a). This approach has been experienced in different industrial contexts and with several meta-models, (Rolland and al. 2004), (Etien and al. 2003b).

Defining gaps based on an As-Is situation described with only one kind of meta-model is not a difficult task. However, it is very unlikely that only one kind of meta-model will be found in a real project. An important issue is therefore to identify gaps in a multi-meta-model environment. Our proposal to address this issue is:

- First, synthesise the As-Is situation using an abstract meta-model
- Second, define gaps for the abstract meta-model

- Third, propagate gaps on the detailed meta-model used in the project

This approach has several advantages: (i) it allows to represent the As-Is situation in a way which is unified and comprehensible by everybody in the project (communication and agreement are thus enhanced), (ii) it allows to start by looking for evolution requirements using only one kind of language, (iii) it allows to ensure that the evolution requirements propagated on the different models are homogenous, and (iv) it allows to ensure that the To-Be system specifications match with the To-Be business models.

The rest of the paper is organised as follows: section 2 presents our approach to model evolution requirements with any kind of meta-model. Section 3 presents the modelling technique that we propose to abstract the different kinds of models that can be found in an IS evolution project. Section 4 shows with the case study of IS change at the Stockholm Central Service company how our approach applies in an industrial environment. Section 5 discusses related works and concludes on the advantages expected from our approach.

EXPRESSING EVOLUTION REQUIREMENTS WITH GAPS

Our approach suggests to specify evolution requirements under the forms of gaps (represented by Δ in Figure 1). In this approach, the As-Is and the To-Be models are both instances of the same meta-model, that we call "specific meta-model" because it has its proper specificities. To each specific meta-model, is associated a language to express evolution requirements. It is composed of gap operators that can be applied on the elements of the specific meta-model. For this reason, these typologies are called "specific typologies of gap operators"

Rather than defining as many specific typologies of gap operators as there are of specific meta-models (UML, i*, KAOS, BPML…), our approach proposes to take a more abstract view that we call generic. A generic meta-model level is thus defined on top of the specific meta-model level. The generic meta-model allows to make explicit the elements and structure of the meta-model that are used to represent the As-Is and To-Be situations.

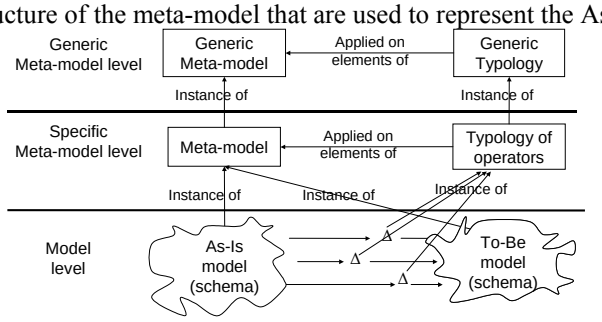

Figure 1: Overview of the gap modelling approach to express evolution requirements

Examples of meta-models at the specific level are Use Case, i*, BPML, KAOS models, etc. For example, gaps can be used to specify the evolution of a software for which some of the requirements are defined with a Use Case diagram. The typology of gap operators that can be used for this purpose will for example include "Add Use Case", "Change Origin of use Case-Actor Association", or "Merge Actors". The Use Case model and typology of Use Case gap operators are designed by respectively instantiating the generic meta-model and the generic gap typology. The remainder of this section presents the generic meta-model and generic gap typology.

Generic meta-model and generic gap typology

According to the generic meta-model, any meta-model is composed of a collection of elements that have properties. As shown in Figure 2, *Elements* are classified into two clusters. First, a distinction between *Simple* and *Compound Elements* is made. Second, elements can be classified into *Link* and *NotLink*. *Compound elements* are decomposable into elements that can be simple or at their turn compound. In particular, any model is a compound element. *Link Elements* are connectors between pairs of elements. Links can be oriented; therefore one of the linked elements plays the role of *Source* and the other of *Target*. In any model, there is a *Root* element. This allows to design minimal models such as the Object class in a class inheritance diagram, or the system boundary in a Use Case Model. Figure 2 shows concepts of the meta-model in white and the types of evolution requirements that apply to them in grey.

For example, a Use Case diagram is a compound element composed of System Boundary, Use Cases, Actors, Associations and Use Case Links. The System Boundary is the Root (it is the minimal content of a Use Case diagram). A Use Case is composed of a primary scenario and several secondary scenarios. It is therefore a compound element with several properties, Priority, Precondition, etc. Associations and Use Case Links are elements of the Link type. Every Association is between an Actor and a Use Case. There are two kinds of Use Case Link, Use and Extend. Both are defined as elements with an is-a relationship to the Use Case Link element. Therefore, as defined for Use Case Link, they connect a source Use Case to a target Use Case.

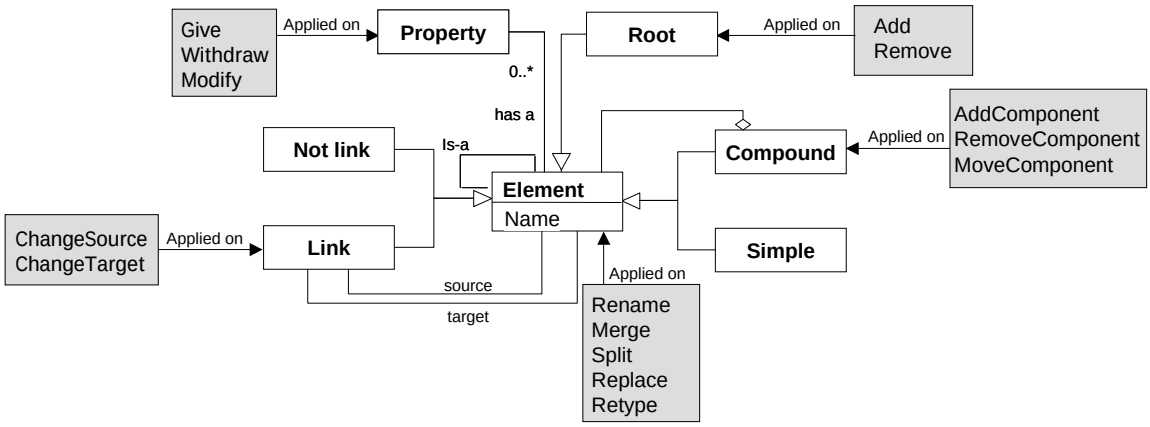

Figure 2: The meta-model for gap typology definition

In our approach, evolution requirements are expressed as gaps under the form of change operations required on models. There are different kinds of such operations: adding or removing elements, changing, replacing, etc. Fourteen operators have been identified and defined on the generic level, i.e. to apply on the objects of the *generic meta-model* (Rolland and al. 2004).

For example, the Rename, Merge, Split Replace and Retype operators can be applied on any kind of element. There also exist operators that can be only applied on a particular type of concept such as like Give, Withdraw, Modify for a Property, Add, Remove for the Root, ChangeSource and ChangeTarget for the Link and finally AddComponent, RemoveComponent and MoveComponent for the compound element.

Developing specific typologies of gap operators

In order to define a specific typology of gap operators for a given specific meta-model (Etien and al. 2003a), it is first necessary to redesign this meta-model. This is done by instantiating the generic meta-model. Then, the generic gap operators are instantiated depending on the type of the specific meta-model objects to obtain the specific gap operators. For example, only the operators Rename, Merge, Split, Replace, Retype, ChangeSource and ChangeTarget can be applied on an association, which is a link between Use Case and Actor. They are thus instantiated. Finally, a domain expert has to prune among the obtained operators those that have no sense for the given specific meta-model.

We have already developed several specific typologies of gap operators, for instance for event-driven business rule models (Rolland and al. 2004), for process model descriptions, for Workflow models and Class inheritance diagrams (Etien and al. 2003b). This experience showed us that not only the generic gap typology approach can be adapted to any kind of method used in a project, but also that the resulting specific typologies of gaps were more complete and provided semantically richer gap operators than the ones developed in an ad hoc way.

ABSTRACTING THE AS-IS WITH GOAL/STRATEGY MAPS

In a multi-meta-model environment, it is possible to define a typology of gap operators for each meta-model. However, it lacks readability insofar as some gaps are defined in a model with or without any link with gaps in another model. In order to remedy to this lack of homogeneity and efficiency, we propose to abstract the As-Is situation using an abstract meta-model, then to define gaps for this meta-model and finally to propagate these gaps on the different meta-models used in the projects.

Our experience shows us that a goal-oriented meta-models and in particular the goal/strategy map formalism allow to adequately abstract a situation by avoiding unnecessary details, and by focusing attention on what is to be achieved (the goals) prior to the ways to achieve it (the strategies).

As shown in Figure 3, a goal/strategy *map* is a directed graph in which nodes are labelled with “goals” and edges labelled with “strategies” (Rolland 2000). An edge enters a node if the strategy that it represents can be used to achieve the corresponding goal, whereas a node is the source of an edge if the achievement of the associated goal is a precondition. Having several edges pointing to the same node allows to represent the multiple strategies available to achieve a unique goal.

A *goal* aims at some situation that an organization wants to see attained. For instance, the goal ‘*Estimate the cost of a reparation service*’ is a goal shown in Figure 3. This goal can be achieved using different strategies. Each of these is supported by the existing Information System.

Strategies define approaches and manners to achieve goals. For example, Figure 3 shows two different strategies to estimate the cost of a reparation service: one is 'without the apparatus', and the second is 'at the customer's'. A reparation cost that is estimated by phone (without apparatus) cannot be very precise. Therefore, the estimated cost is a fork and should be recorded as such in the system. On the contrary, the purpose of going at the customer's place is to have an exact estimation. Therefore, the system supports this strategy by offering adequate features that also records exact estimations.

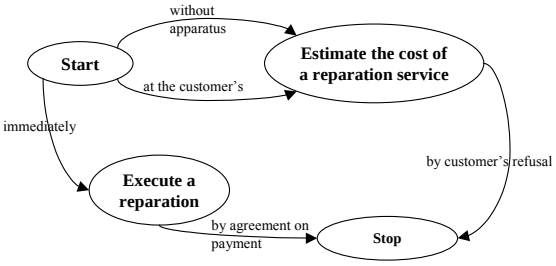

Figure 3: Example of a map.

The two goals of the map presented in Figure 3 are enacted in a non deterministic way: reparation can be executed without estimation, and the other way round, an estimation of the cost of a reparation service can be done without any reparation. Besides, there is no specific ordering between each activity. For a given customer asking for a single service, the reparation can occur before, after, or at the same time as the estimation. However, there can be no agreement on payment or customer's refusal if the reparation has not been undertaken or the evaluation achieved, respectively.

Sections of a map (i.e. triplet <source goal, target goal, strategy>) can be detailed using maps based on a refinement mechanism. According to this mechanism, it is possible to refine a section of a map at level i into an entire map at a lower level i+1. This allows to view a goal together with its strategy as a complex graph of sub-goals and their associated strategies.

While changes that correspond to completely new requirements can easily be propagated from the abstract model onto the other models by looking at how the new goal is implemented into a new software component (Landtsheer and al. 2003), there is no systematic way to control the propagation of the other kinds of evolution requirements in the abstract model onto the other models. In order to allow such propagation or in other terms to control co-evolution between the goal level and the business or the architecture level, it is necessary to define some patterns or some rules based on the elements of the different meta-model. For example a section of the goal/strategy formalism is represented in the EBML model by a process sequence. It is thus possible to define some rules to propagate the merge of two sections onto the EBML model. These rules remain for the moment model dependant and based on the experience of the change engineer. Further works are needed to design pattern formalising these rules.

CASE STUDY: IS EVOLUTION AT SCS

SCS (Stockholm Central Service) is a workshop offering overhaul and reparation services for home electronics. This includes mostly audio-video products, but SCS also offers reparation of products such as computers, refrigerators, and washing machines. It was decided to re-organise the company in order to solve a number of issues relating to productivity. The project to change SCS business was undertaken with the aim of improving competitiveness, suppress insufficient response times to customers, and find an appropriate organization of the work structure.

Several different meta-models have already been used to model the As-Is situation. Business processes were described using the Extended Business Modelling Language (EBML). The legacy IS was itself designed using Entity/Relationship (E/R) diagrams for the initial modelling of the data structures, and Class Diagrams (using UML) for advanced specifications of the Business Rules.

Overview of the company

SCS's customers consist of resellers, suppliers and private persons. The most important clients are contracted clients, mainly resellers and suppliers, which constitute 90% of the total number of customers. The range of services is the same for the three different types of customers. However, SCS categorises customers with respect to price settings, conditions and types of contracts.

A reparation service requested by a client is registered as an order. An order is categorised either as a service order or an external service order. The type of order is determined by the way the apparatus is received and delivered, by the kind of reparation required, and by the client category. A service order comprises all three

categories of customers and involves the customer delivering the apparatus by himself. The reparation takes place at the workshop. An external service order comprises only private persons and involves an external SCS service delivery at the site of the client's home, based on an initial appointment. The reparation can take place either directly at the client's home or at the workshop.

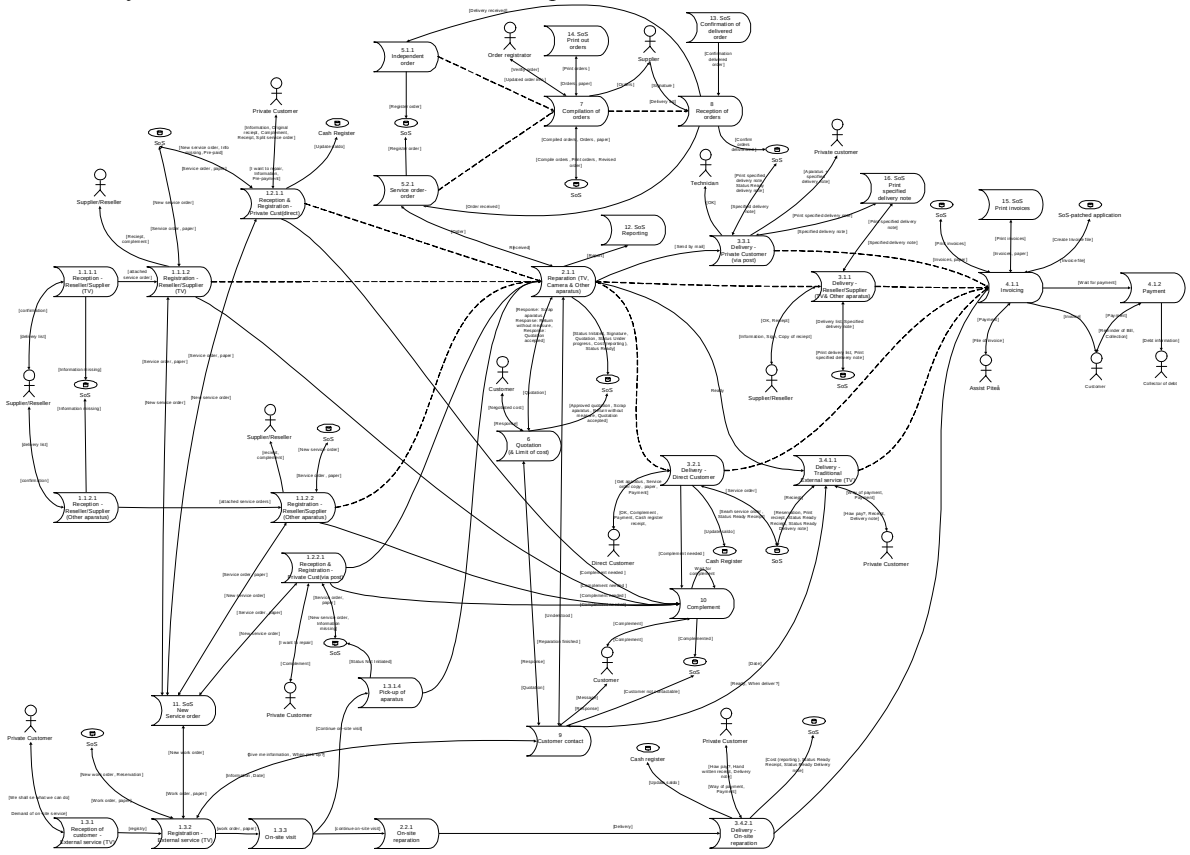

Figure 4: Extract of the As-Is key BP model specified with EBML[1].

Order processing is the most important business process at SCS. It includes: reception of apparatus, reparation of apparatus, delivery of apparatus, and invoicing of service. Each activity has several individual variations, depending on the type of reception and delivery, reparation and customer.

The processes of reception and delivery of apparatus, take place at different locations in the workshop. The reception of private persons is situated in one of the two buildings of the workshop, while the reception of suppliers and resellers, delivering apparatus by trucks, is situated in the other building.

The company does not use the information system supporting the process order to its full potential. In parallel with the computerised information system, they use a manual folder system, where they keep track of all their orders and delivery documents.

The SCS order processing process is rich and complex: it involves a lot of secondary processes (the most important ones are about 20), many variations (for example there are five ways to receive the apparatus), and many interactions between processes (this is illustrated by the arrows in the business process model shown in Figure 4).

Figure 5 illustrates the As-Is EBML process Reparation. Due to readability reasons, the labels for each process entity were changed to numbers. For the sake of space, Figure 5 only visualises the control flow of the reparation process, i.e. the ordering of messages received and sent (which represents the interaction with other actors and processes), and the ordering of the process activities. More specifically, the SCS reparation process is characterised by three types of reparations indicated by the three different process sequences following the first decision point (no. 4 in the Figure). The reparation activity is then followed by a number of messages, activities and other decision points. These put conditions on whether the responsible engineer has made a quotation, signed and updated the status of the service order in the IS or not. Depending on the engineer, this routine is not

[1] This picture cannot be fully shown here for sake of space. More readable versions can be accessed at URL at http://crinfo.univ-paris1.fr/users/benachour/ACIS/

done at different points in time. This is indicated in the As-Is EBML model by the loop from decision point 20 to box 3.

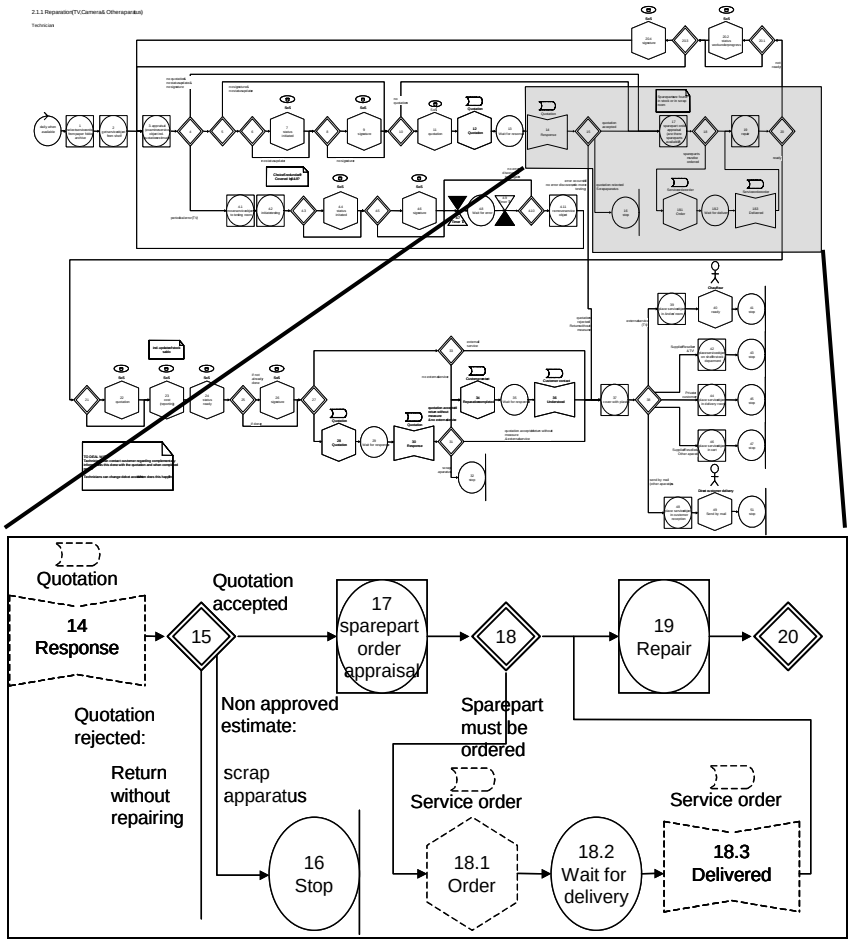

Figure 5: The As-Is *Reparation* EBML process model[2] and a detail. Diamonds are decision points, circles are activities, and figures in dotted line indicate the existence of a sub-process.

Abstracting the As-Is situation with maps

Several maps were developed to specify the synthetic and unified view of the As-Is situation at SCS. These maps belong to different levels of abstraction/detail. The two following subsections present two maps that belong to the two highest levels of the resulting hierarchy. At level 1, a single goal/strategy map defines the general business goals and strategies of the company. At level 2, the presented map illustrates the processing of the customer service requests in SCS's offices.

Other maps were designed on more detailed levels, for example, at level 3, maps identify the services provided by the system. For example, one of the maps indicates that the system provides a feature supporting reparation service cost estimation. At level 4, the maps specify the user interactions with the system. This level is the lowest level dealt with here because the map presented is composed of goals and strategies that correspond to individual activities that cannot be further decomposed without entering into the internal details of the system.

Level 1: provide reparation service

The goal/strategy map shown in Figure 6 presents a general overview of the organisation and can be commented as follows:

Get closer to the market: The notion of market represents the competitive environment of SCS, which has to be scanned and analysed in order for the business to survive, i.e. to actively approach the market and intervene with it. Approaching the market is realised by two main strategies, where analysing the market demand

[2] This picture cannot be fully shown here for sake of space. More readable versions can be accessed at URL at http://crinfo.univ-paris1.fr/users/benachour/ACIS/

complements and supports the strategy of integrating the new demand into the business. The knowledge that the analysis brings forward facilitates for the business to smoothly approach the market.

Establish a contract: The most important customers (90%) are contracted customers, which at present mainly are represented by resellers or suppliers. The contracts are considered as issued when a new customer has signed a new contract or when contracts are reissued, either by prolongation, by adding new services to the former contract or by deleting services in a former contract.

Manage the customer relationship: It is considered that the relationships with contracted customers and with customers that demand services without contracts should be managed in order to develop customer loyalty. Customer relationship management at this level is realised by the processing of customer requests, which are either informational requests or requests for services offered at SCS. Moreover, the customer relationship management includes control of the flows between SCS and its clients, i.e. information, contracts, requests and communication where SCS has the role of the service provider. Indeed, it is also important to point out the control of the flows between SCS and its suppliers, where SCS has the role of the client and therefore need to maintain good relations with its own suppliers.

Stop: The relation with the customer is considered terminated either by the termination of contracts, by the expiration of contracts or by finishing a service requested by a non-contracted customer.

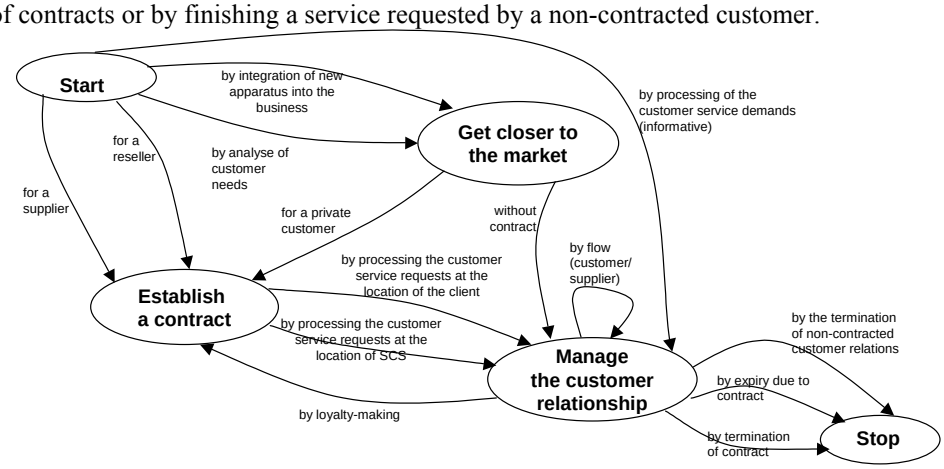

Figure 6: Provide reparation service high-level goals and strategies

The goal/strategy map shows that once a contract has been established with a customer, the relationship with this customer can be managed in two different ways: by processing the customer's service requests at the customer's location, or by processing the service requests at SCS. The next subsection describes in more details how this second strategy is achieved in the current situation.

Level 2: Manage customer relationship by processing the customer service request at the location of SCS

The Map in Figure 7 represents the service of processing customer requests at the site of SCS. It illustrates the three main goals and their related strategies, which correspond to the activities at the heart of the business of SCS, i.e. repairing home electronics or so-called audio-video products.

The goals are the following:

Stock an apparatus: SCS offers four different alternatives for stocking the apparatus, each categorised by type of customer, reparation and delivery. The apparatus are received either by the delivery of units from resellers, by individual deposition by the customer, by post or by SCS external home service.

Estimate the cost of a reparation service: The service requests may include a request for a quotation before the reparation takes place. Quotations are managed by either three ways; consulting the client by telephone, by examination of the apparatus either at SCS or at the customer's, depending on if it's an internal or external service request.

Execute a reparation: The execution of a reparation process is the actual realisation of the customer service request, managed by a technician who is either a specialist on particular apparatus or brands or a generalist, who repairs different types of apparatus. Depending on the type of apparatus and the problem identified the reparation is managed either directly or instantly, by a prerequisite test or by several stages without the initiating and prerequisite test. These three strategies are the ones embedded in the process model shown in Figure 5.

Stop: The processing of customer requests and reparation processes are considered terminated by the return of the repaired apparatus to its owner, which is done in ways that correspond to the four different ways they are received. The requests may also be terminated at earlier stages in the process of processing customer requests at

SCS. The customer may choose to refuse to the quotation offered, which in case the reparation has not yet started ends the process at the point of refusal, or which in the case the reparation has already started may imply that the apparatus is either returned or disposed.

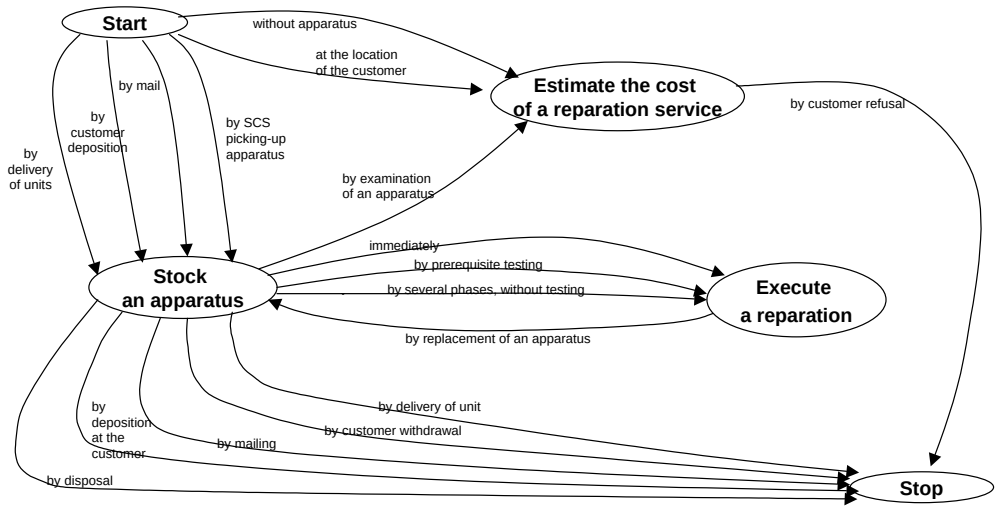

Figure 7: Goal and strategies governing the management of the customer relationship by processing the customer service request at the location of SCS

Examples of evolution requirements expressed as goal/strategy gaps

The evolution requirements were first analysed under the form of gaps with the As-Is goal/strategy models. Numerous gaps could be found at this levels; Table 1 provides some of those that were found on the customer-relationship management goal/strategy map shown in Figure 7.

	Goals	Strategies
Merge	1. *Estimate the cost of a reparation service* and *Execute a reparation* into *Repair an apparatus*	1. *without apparatus* and *at the customer* into *without stocking*
		2. *by delivery of units* and *by customer deposition* into *collective stocking*
		3. *by mail, by SCS picking-up apparatus* and *by customer deposition* into *individual stocking*
		4. *immediately, by prerequisite testing, by several phases without testing* and *by replacement of an apparatus* into *by prerequisite quotation*
		5. *by delivery of unit* and *by customer withdrawal* into *collective stocking*
		6. *by mail , by delivery to customer* and *by customer withdrawal* into *individual stocking*
Split	1. *Stock an apparatus* into *Receive an Apparatus* and *Return an apparatus*	
Change origin		1. *by disposal* source goal to *Repair an apparatus*
		2. *without stocking* target goal to *Receive apparatus*
		3. *collective stocking* target goal to *Receive apparatus*
		4. *individual stocking* target goal to *Receive apparatus*
		5. *without stocking* target goal to *Return apparatus*
		6. *collective stocking* target goal to *Return apparatus*
		7. *individual stocking* target goal to *Return apparatus*
Remove		1. *by examination of an apparatus*
		2. *by customer refusal*
Add		1. *by invoice*

Table 1: Examples of evolution requirements found on the goals and strategies of the Manage the customer relationship by processing the customer service requests at SCS map

The table shows that each of the As-Is goals has been either merged or split. This relatively radical change is due to the underlying requirement to strengthen the process-oriented organisation (Davenport 2000), (Hammer 1996) at SCS. More specifically, in order to change the current focus on separate functions such as stocking and repairing towards a process-oriented organisation, the goal *Stock an apparatus* was split into two new goals, *Receive* and *Return an apparatus*. This split was also a result of the need to change radically from ineffective stocking strategies to more rationale ones.

Furthermore, the As-Is Map defined cost estimation and execution in two individual goals: *Execute a reparation* and *Estimate the cost of a reparation service*. The corresponding business processes were indeed separate, and as a result the customer could reject a quotation although the reparation was already started or even finished. Merging these goals into *Repair an apparatus* identifies the interdependency of the two. The underlying decision was to merge the corresponding business processes instead of keeping them independent. Therefore, the strategy *by prerequisite quotation* was introduced as the only strategy to achieve this goal. This gap identifies that the underlying To-Be process should avoid the ill-defined ordering of reparation and quotation

handling, which caused interrupted and in the end costly reparations. The impact of these gaps on the changes to perform on the current business processes is multiple. First, two well delineated fragments of process should be defined, one for quotation and the other for reparation. Second, the links between those should be changed: there should not be any loop from the latter to the former, and there should be no way to reach the latter without passing by the former.

Two steps were undertaken to ensure a consistent definition of the evolution requirements impacting apparatus reception and apparatus return. First, the ineffective strategies for stocking apparatus were merged into the strategies *collective, individual and without stocking*. In the second step, their origins were changed in order to correspond to the new goals as indicated in table 1.

Another important aspect of the gaps reported here is the introduction of the termination strategy *by invoice*. Indeed, it appeared that in the current situations not all reparations ended up in a payment. On the contrary, the To-Be Map identifies that if the customer rejects the quotation, then no repair is achieved and no payment is required. If the quotation is accepted and the reparation is realised, then the invoice should be delivered. The underlying business processes should reflect these constraints by tightening the links between these three aspects of the business.

Propagating goal/strategy gaps on the BPML, E/R and Class Diagram models

The impact of the gaps found at the goals-strategy level onto the EBML models, E/R diagrams and Class diagrams, is multiple.

Just to give an idea, the merge of the goal *Estimate the cost of a reparation service* with *Execute a reparation* into the new goal *Repair an apparatus*, and the introduction of the new strategy *by prerequisite quotation* in the goal-strategy map caused the removal of 7 decision points, 4 send messages and 1 whole process sequence from the EBML models.

By removing the decision points (no. 4, 4.3, 4.5, 5, 10, 20.1, 20.3 in Figure 6), the message sending (no. 4.4, 4.6, 20.4, 22) and the process sequences (no. < 25; 26; 27; 28; 29; 30; 31; 32>), the processes for quotation handling and reparation respectively became strictly sequential. The new strategy *by prerequisite quotation* now requires a quotation before the reparation can take place. This allows consistent quotation handling and eliminates interrupted reparation processes due to customer quotation refusal.

The evolution requirements reported here as gaps with the As-Is EBML models had not been identified initially when only EBML models, E/R diagrams and Class diagrams were available. This has an impact on the future enterprise organisation, but also on the design of the future Information System.

CONCLUSIONS AND RELATED WORKS

Information Systems evolution is a topic that can be dealt in different ways depending on the stressed perspective. Indeed, three types of evolution can be defined (Swanson 1976) (1) adaptive: adding new features, (2) corrective: fixing faults, and (3) perfective: enhancing performance.

In (Reichert and al. 1998), the authors propose to change the model only for one instance or one type of problem. IS so evolves in order to manage situations that were not foreseen. These non-permanent evolutions are performed through operators that are very simple and mainly correspond to add or remove task or message.

Other approaches as (Banerjee and al. 1987), (Delgado and al. 2002) also focus on one level and use operators to define permanent, adaptive evolution. The set of operators is each time complete but remains semantically poor in the sense that there does not exist merge, split or even replace operators. These authors propose rules and invariants that allow to define consistent operators. However, these rules and invariants only manage consistency at one level (in these cases database schema) and do not permit evolution between different models.

(Terrasse and al. 2002) manages evolution of Information System using several meta-models. However, it is not the propagation of the evolution from a meta-model to another that is studied but the evolution between meta-models themselves. This approach is independent of the chosen meta-models and is based, as ours, on a generic meta-model called generic modelling paradigm, which corresponds to the standard UML semantics.

Some authors as Landtsheer and al. (2003) propose to propagate evolution occurred on one meta-model onto other meta-models. Thus, they derive event-based specifications, written in the SCR tabular language, from operational specifications built with the KAOS goal-oriented method. This approach is based on (i) a mapping between elements of the SCR language and elements of the KAOS method (2) transformation rules, which resolve semantic, structural or syntactic differences between the two models. However, this approach only manages incremental evolution, which is just one type of adaptation.

Our approach promotes a multi-meta-model environment to deal with a system evolution project. For this purpose, the approach allows (i) to specify a typology of gap operators for any kind of meta-model; (ii) to focus on evolution requirements what allows to express the requirements evolutions implied by the organisational business expectations and the needs for the future; (iii) to develop a more complete view of the As-Is and To-Be situations. Whereas each of the meta-models initially used focussed on a specific aspect (business processes, information structures, business rules, etc), the goal-oriented view tackles an abstract level at which an overall vision of the 'business-IS' couple; and (iv) to propagate the goal/strategy evolutions onto others models. However, this last point contrarily to (Lantsheer and al. 2003) allows to manage any kind of evolution and not just incrementations. However, it remains, for the moment based on the experience of the engineer. The definition of patterns that deal with this issue is the subject of our current research.

REFERENCES

Banerjee, J., Kim, W., Kim, H.-J., Korth, H. F. (1987): *Semantics and Implementation of Schema Evolution in Object Oriented Databases* In Proc. of the ACM-SIGMOD, pages 311-322, USA, May 1987.

Davenport, T. H. (2000): *Mission Critical*. Harvard College, USA.

Delgado, C., Samos, J., Torres, M. (2003): Primitive Operations for Schema Evolution in ODMG Databases Proceedings of OOIS03, Geneva, Switzerland

Hammer, M. (1996): *Beyond Reengineering*. Harper Collins Publishers, USA.

Etien, A., Denecker, R., Salinesi, C. (2003): *Extending Methods to Express Change Requirements*. In Proceedings of EMSISE'03, Geneva, Switzerland

Etien, A., and Salinesi C. (2003). *Towards a Systematic Definition of Requirements for Software Evolution: A Case-study Driven Investigation,* In Proceedings of EMMSAD, Velden, Austria.

Jarke, M., Pohl, K. (1994): *Requirements Engineering in 2001: Managing a Changing Reality*. IEEE Software Engineering Journal, pp. 257-266. November.

de Landtsheer, R., Letier, E., van Lamsweerde, A.. (2003): *Deriving Tabular Event-Based Specifications from Goal-Oriented Requirements Models.* Proceedings of RE'03, USA.

Pettersson, P., Wäyrynen, J. (2001): *Verksamhetsöversikt – Stockholm Central Service AB*. University of Stockholm/Royal Institute of Technology, Internal report, Sweden.

Reichert, M., Hensinger, C. and Dadam P. (1998): *Supporting Adaptive Workflows in Advanced Application Environments*. Proc. EDBT Workshop on Workflow Management Systems, Valencia, pp. 100 - 109.

Rolland C. (2000): Intention Driven Component Reuse. *Information Systems Engineering* (S. Brinkkemper, E. Lindencrona, A. Solvberg, eds). Springer, pp.197, 208.

Rolland, C., Salinesi, C., Etien, A.(2004): *Eliciting Gaps in Requirements Change*. Requirements Engineering Journal, Vol. 9, pp1-15.

Salinesi, C., Rolland, C. (2003): F*itting Business Models to Systems Functionality Exploring the Fitness Relationship*. Proceedings of CAiSE'03, Klagenfurt./Velden, Austria

Swanson, E. B. (1976): *The dimensions of maintenance*. Proceedings of 2nd Conf. on Software Engineering, pages 492-497, San Francisco, USA.

Terrasse, M.N., Savonnet, M., Becker, G., Leclercq, E. (2003): *UML-Based metamodeling for Information System Engineering and Evolution*. Proceedings of OOIS03, Geneva, Switzerland

COPYRIGHT