Compliance Gaps: A Requirements Elicitation Approach in the Context of System Evolution

Camille Salinesi and Anne Etien

C.R.I.- Université Paris 1 – Sorbonne
90, rue de Tolbiac, 75013 Paris, France
Tel: 00 33 1 44 07 86 34
Fax: 00 33 1 44 07 89 54
{camille,aetien}@univ-paris1.fr

Abstract. Eliciting change requirements in the context of system evolution is different from eliciting requirements for a system developed from scratch. Indeed, there is a system and documentation that should be referred to. Therefore, the issue is not only to identify new functions, but to uncover and understand differences with the current situation. There is few approaches that systematise the identification and documentation of such change requirements. Our approach is based on the analysis at the model level of the fitness relationship between the business and the system. Our experience showed us that another kind of change requirements could also be found when asking the question of continuity at the instance level. The literature already proposes so called "modification policies" that allow to manage current instances of the system workflows and business processes when their model evolve. However, these approaches are not interested in the elicitation of the requirements that relate to these modification policies, but to the technical solutions that these policies provide. The position taken in this paper is that change requirements can be elicited by analysing system evolutions through modification policies at the instance level. The paper proposes to document these requirements using the same approach as other change requirements.

1 Introduction

Organisations experience today more than ever changes in their environment and in their business. Not only they must adapt to these changes, but their software must evolve too. Indeed, to have a software that fits the needs of a business, it is necessary that its evolution matches the business evolution [1].

Evolution creates a movement from an existing situation to a new one. Change is thus often perceived as the transition from the current to a future situation, respectively defined by As-Is and To-Be models [2]. We already showed that rather than building To-Be models, it is more efficient to elicit change requirements by specifying *gaps* that state what differentiates the future situation from the current one. Each situation being defined with models, it is for example possible to specify that elements have to be added, removed, merged or split. Change requirements are then documented using a generic typology of gaps [3] that can be used with any meta-model, as shown in [4].

D. Konstantas et al. (Eds.): OOIS 2003, LNCS 2817, pp. 71–82, 2003.

Our experience showed us that guidance of the requirements elicitation process in the context of software evolution is needed. For example, we proposed in [3] an approach that uses the evolution of business processes to uncover system change requirements. The purpose of this paper is to propose such guidance by reasoning on the consequences that high level change requirements have at the instance level, i.e. on the running system and business process instances. Indeed, when a change is required, the question of what happens with the current instances of the corresponding processes, classes, etc. can be raised. Several policies that solve this issue at the technical level have already been developed [5], [6], [7]. The principle of these *modification policies* is to specify what should technically be done at the instance level to reach a situation that complies with the To-Be models. These policies are useful to reason on the software and business at the instance level. However, they miss the important point of expressing at the requirement level what has to be done.

The position taken in this paper is that adopting modification policies implies new software change requirements that can also be defined with gaps called *compliance gaps*. As a result, our approach proposes to elicit gaps in a number of ways: (i) by analysing business evolutions, (ii) by looking at the consequences of these evolutions on the system (this includes system evolutions at the model level as well as modification policies on the instance level), and (iii) by transposing on the model level the requirements identified on the instance level. The remainder of the paper is structured as follows: section 2 presents the approach by introducing a typology of modification policies and suggesting the corresponding compliance gaps. An example is developed in section 3; section 4 discusses related works; section 5 evokes future work in our research agenda.

2 Presentation of the Approach

2.1 The Context

We adopt the change handling view in which change creates a movement from an existing situation captured in *As-Is models* to a new one captured in *To-Be models* [8]. *As-Is* models describe the current situation whereas *To-Be* models describes the future. We believe that software evolution should be *gap* driven: gaps express what has to changed/adapted to the new situation. Therefore, a collection of gaps documents change requirements. Our approach distinguishes two types of gaps: business gaps and compliance gaps. Both of them express change requirements, but their origin is different.

Business gaps are system change requirements that fit to the need of business evolutions. In a former paper [3] we proposed to define business gaps using operators that express the transformation of models of the current situation into the future one. A generic gap typology that can be used for any meta-model has been defined.

Compliance gaps define at the requirement level the impact expected from modification policies at the instance level. Therefore, these gaps tell how to ensure continuity when putting into practice the business gaps. Modification policies offer t technical solutions to support continuity during the As-Is / To-Be transition. It is for example possible to *migrate* an instance of the As-Is model to an instance of a To-Be model i.e. consider that the As-Is model instance is an instance of the To-Be model

too. Another possibility is to let the As-Is instance terminate its lifecycle according to the As-Is model. The current instance can also be removed, or abandoned in the case of a process, etc. The choice of modification policies creates new requirements that can be considered as changes, and therefore be modelled at their turn using gaps. We believe that these gaps can be expressed in the same way as business gaps, e.g. using a gap typology [3].

Figure 1 summarizes our gap elicitation approach. The figure indicates that the central issue is the transition from the As-is to the To-Be situation. Two levels of analysis are emphasized: the model level and the instance level. On the model level, change requirements are expressed by gaps with the As-Is models. These requirements are of two kinds: business change requirements and compliance requirements. Business change requirement produce business gaps, while compliance requirements produce compliance gaps. In our proposal, compliance gaps are elicited by analysing the choice of modification policies when business gaps are put into practice on the instance level.

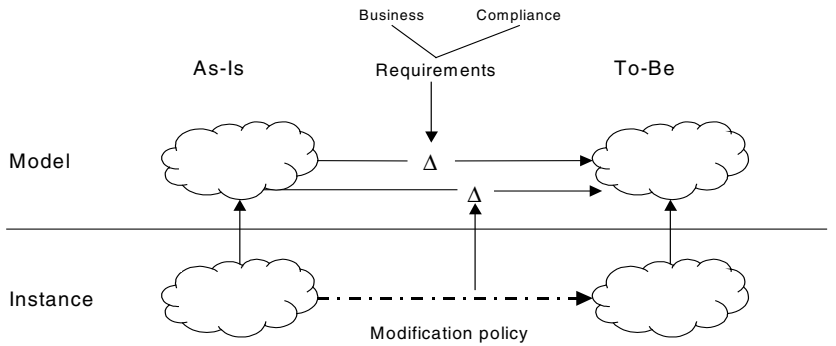

Fig. 1. Overall schema of the approach

The collections of modification policies proposed by literature [5], [6], [7], [9], [10] [11], [12] are very similar the ones to the others. The next section presents a collection of modification policies adapted from [5].

2.2 Presentation of the Modification Policies

Five modification policies are proposed in [5]: Flush, Abort, Migrate, Adapt and Build.

- *Flush*: in this policy, the lifecycle of the As-Is model instances is terminated according to the As-Is model. Therefore, To-Be requirements do not affect the current instances of the As-Is models and new instances are created using the To-Be models. As a consequence, the change is only seen when new instances of the model are started (e.g. for new customers, new products, or when new contracts are made, etc). The current instances (existing customers and products, or contracts that are currently being managed) do not benefit from the required evolution.
- *Abort*: in this case, the current instances are abandoned (e.g. booking requests are cancelled). This policy may generate losses to the organisation, which in some cases may be unacceptable. Its purpose is to start on a standard basis where all

instances comply to the same model. Usually, the aborted instances are re-built from scratch using the To-Be models.

- *Migrate*: the principle of this policy is to transfer current instances of the As-Is models onto the To-Be models so that they directly benefit from the changes. Migration is implemented by compensation procedures that transform the current model instances so that they instantiate the To-Be models too. For example if a new structure is defined for contracts, a migration policy will transform all the existing contracts so that they implement the new structure. If a booking process changes, then all the existing bookings can be migrated so that they terminate according to the new model.
- *Adapt*: this policy is designed to manage cases of errors and exceptions. Its principle is to treat some instances differently because of unforeseen circumstances.
- *Build*: this policy handles the construction of a new model from scratch. It can for instance be used when a component is introduced in an IS or when processes for a new business are adopted in a company.

Our typology of modification policies, shown on Fig. 2, does not include the Adapt and Build policies. Indeed, these policies are not compliant with the framework that we defined for our approach (see Fig. 1). On the one hand, the Build policy assumes that there is no As-Is model. On the other hand, the Adapt policy assumes that the As-Is model does not include the evolving elements. On the contrary, our approach assumes that business change requirements as well as the compliance requirements are specified based on a complete and consistent model of the As-Is situation.

Our typology divides the modification policies in three groups: migrate, flush and abort. Each of these groups includes several sub-policies. In all policies belonging to the *migrate* group, instances begin their lifecycle with a model and finish it according to another. All the *flush* policies allow the instances of As-Is models to continue their lifecycle as defined in As-Is. Finally, all *abort* policies allow to undo one or several steps in the lifecycle of As-Is instances, or even to completely abandon these instances.

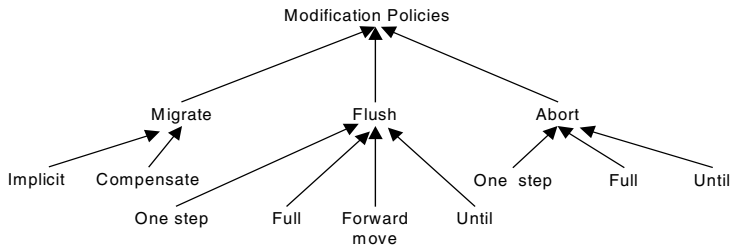

Fig. 2. Modification policies typology

Each policy leads to a specific reasoning on what should be done to ensure a consistent transition from As-Is instances to To-Be instances. Different requirements can be suggested for each of the proposed policy. A number of proposals are for instance made in the remainder of this section.

Implicit migration: Fig. 3 shows that As-Is instances can also instantiates the To-Be models. If all preconditions to proceed as defined in the To-Be are fulfilled, then the instances can terminate their lifecycle according to the To-Be models. Such

implicit migration can for instance occur when a process is replaced by a new one with the same beginning. Instances of the As-Is process model which status corresponds to the common part of the As-Is and To-Be model can proceed either according to the As-Is model or according to the To-Be. The choice has to be defined in a compliance gap.

Migration by compensation: this policy is useful for As-Is instances that are in a state without equivalent in the To-Be models. Compensation consists in transforming the instances so that they match the To-Be models. It can for instance be used when a product structure modification occurs: rather than re-developing all existing products, compensation can be used to adapt them so that they fit the new structure defined in the To-Be models. The corresponding requirements can be simply added into the requirements specification using compliance gaps.

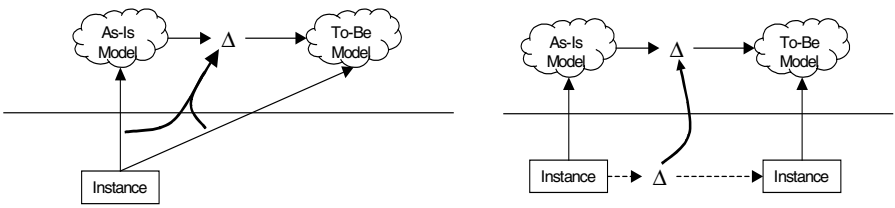

Fig. 3. Implicit migration policy (left), Compensation migration (right)

In all of the four flush policies, the lifecycle of existing instances proceeds as defined in As-Is models. This can hold for one or several stages or even until the lifecycle is completed. A number of things are required to achieve a flush policy : (i) elements of the As-Is models must be introduced into the To-Be models, so that they define how to flush; if these elements were removed or replaced, compliance gaps must be specified to re-introduce them, and (ii) compliance gaps should also specify how to check that there is no new instance of the re-introduced elements. Besides, one step flush, until flush and forward move flush can be combined with another modification policy. Their purpose is then to reach a state in which one can decide upon how to terminate the instance lifecycle.

One-step flush: As Fig. 4 shows, this policy is useful to deal with As-Is instances that are in an unstable state, i.e. for which no migration can immediately be achieved. The requirement is then to reach a stable state before migrating. This is for instance the case of on-line purchase transactions for which it is preferable to terminate in-progress purchases to decide upon how to proceed with the sales process.

The issue raised by the **until flush** policy is similar to the one raised by the next step flush policy. However, in the case of the until flush, a specific state that has to be reached using the As-Is model is chosen in advance. In the purchase example, one can decide that it is only once the purchase transaction is in the “committed” state (and only then), that the migration can be achieved. If for example the purchase transaction is suspended, the As-Is model is still used when the transaction is resumed.

Full flush: in this policy, instances of the As-Is models are never migrated towards the To-Be model. If the To-be exploits persistent information such as object histories, or process traces, then compliance gaps must be introduced to specify how that information can be maintained and retrieved with the As-Is models.

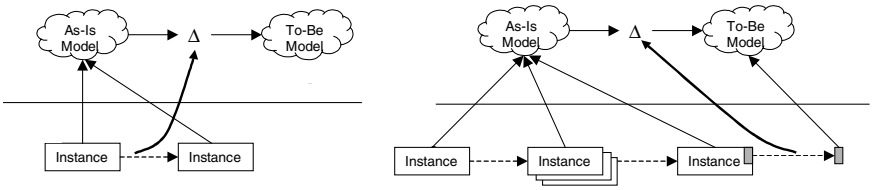

Fig. 4. One step flush policy (left), Full flush policy (right)

Let's take the example of a contract management process. When this process is changed, it can be decided that current contracts should still be managed As-Is, while all new contracts should apply the To-Be model. However, contract management includes risk analysis which is based on the knowledge of bad payers. This information appears in the trace of payment reception and bone of contention activities. Compliance gaps should be specified in To-Be models to integrate the exploitation of this trace information.

Forward move flush: some situations can be not enough informative to decide on how to proceed to comply with the To-Be models. Then, it is necessary to look at what happens in the next step before migrating, flushing completely or aborting. Fig. 5 shows that the purpose of this policy is not to reach a different state, but to find out what direction is about to be taken. The corresponding compliance gaps specify how to catch the move direction, and decide on how to proceed. Let's take the example of an evolving contract-making process. Proposals have already been made to customers in the As-Is way. It is decided to wait for the next event to decide upon how to proceed. If the customer takes contact to answer on the proposal, the salesman proceeds according to the To-Be models. If the customer never calls, then a timeout triggers the As-Is proposal cancellation procedures.

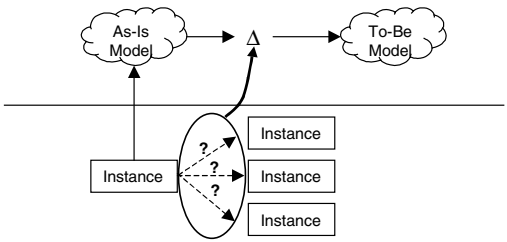

Fig. 5. Forward move flush policy

Full Abort: as Fig. 6 shows, it is possible to remove/cancel/abort/rollback current instances of an As-Is model, e.g. when their value is considered insufficient to justify the cost of a migration. Compliance gaps can however be defined to indicate how to resume with the To-Be model. For example, in the case of a library, it can be decided to remove all pending booking requests because the booking process and system are changing. However, compliance with the new process is ensured by contacting the borrowers whose requests were cancelled, and suggest them to resubmit their request using the new system.

Rather than the radical abort, it can be decided to backtrack so as to reach a start that is exploitable to ensure the transition to the To-Be model. Like for the flush policy, these abort policies can be one-step or until a specific state.

One-step abort: As shown in Fig. 6, the purpose of this policy is to find back in the history of As-Is model instances a state that can be exploited in the To-Be. Compliance gaps relating to this policy should indicate how to achieve the abort and how to proceed once the abort is achieved. Let's take the example of a Bank in which it is decided to offer a new way to calculate the account balance: rather than achieving an imperfect day-to-day calculation based on incomplete information, it is decided to propose a batch procedure that compiles all the account movements at the end of each month. When a customer selects the new procedure for his/her account, all the calculations achieved since the beginning of the month are aborted. This one-step abort allows an immediate enactment of the customer choice.

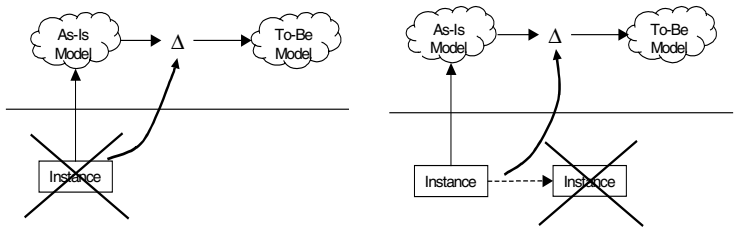

Fig. 6. Full abort policy (left), One step abort policy (right)

To summarize the proposed approach :

(i) business change requirements generate business gaps,
(ii) it is proposed to analyse the impact of business gaps onto existing instances of the As-Is models,
(iii) the analysis is driven by the choice of a modification policy,
(iv) compliance gaps are defined to specify at the model level the requirements linked to the chosen modification policy

The next section illustrates this approach on the case study of a hotel room booking system.

3 Case Study

A system handles room booking for several hotels in a centralised way. A project is undertaken to change the hotel booking business process in order to improve competitiveness. Rather than performing a detailed and complete analysis of the new system, it was decided to identify and specify its gap with the current system [13]. Based on a first-cut business gap model highlighting the new business requirements, compliance gaps were searched for.

The main service provided by the system is the support of the sales processes. This involves several goals: to "Construct a product list" (sales concerns simple products such as single room, double room, or double twin) and to "Manage booking contracts". This situation is modelled in Fig. 7 with the MAP formalism [14] [15]. A *map* is an oriented graph which nodes are *goals* and links s*trategies*, i.e. ways to

achieve a goal. For example, Fig. 7 shows that there are two ways to "manage booking contracts" once the product list is constructed: directly at the hotel desk with the "on the spot strategy" or "by a third party", i.e. a travel agency or a tourism information office.

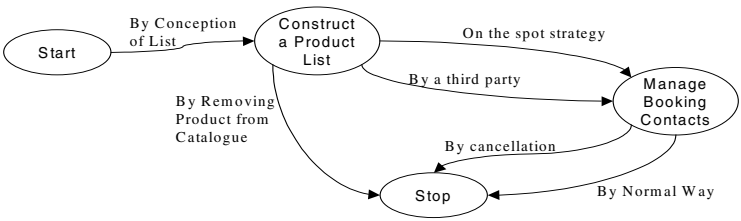

Fig. 7. The As-Is Map: Sell hotel products

In the current situation, products are independently designed in a flat list which is augmented each time a new product is conceived. Once products are in the list, they are used for contracts. Any product of this list can be removed to achieve its lifecycle. Booking contracts are either created "on the spot" in a hotel or "by a third party". The contract management process ends up either by cancellation of the contract, or by consumption of the associated product by the consumer. Therefore, the current situation is strongly product oriented.

A number of evolutions were required. Three major evolutions can be highlighted: (i) from now, the system should be customer centric; (ii) it should be possible to propose to customers complex products such as packages including tourist activities; (iii) The sales channels have to be diversified.

For each of these evolutions, business requirements were specified under the form of gaps with the As-Is Map of Fig. 7. Some of these are shown in Table 1.

Table 1. Example of gaps between the As-Is and the To-Be models

Code	Operator	Element
RNS1-2	Rename	*By conception of list* into *By conception of product catalogue*
RNS2-1	Rename	*By a third party* into *By offering booking facilities to customer by a third party*
RNS3-1	Rename	*On the spot* into *By offering booking facilities to customer on the spot*
AI1-1	Add	*Attract people*
AS1-1/2	Add	*By Promotion,*
AS2-1/2	Add	*By marketing,*
AS3-1	Add	*By exclusion,*
AS4-1	Add	*By tracing non satisfied customers*
AS5-1	Add	*By managing customer's information*
AS6-3	Add	*By offering booking facilities to customer by a web site*
AS7-1	Add	*By keeping customer's loyalty*
RPI1-2	Replace	*Construct a product list* by *Offer a product catalogue*
RPI2-1	Replace	*Manage booking contract by Manage customer's relationship*
RPS3-1	Replace	*By cancellation* by *By cancelling booking*
RMS1-1	Remove	*By normal way*
COS1-1	Change Origin	*By cancelling booking* source intention to *Manage customer relationship*
COS2-1	Change Origin	*By cancelling booking* target intention to *Manage customer relationship*

Table 1 shows that the formulation of all As-Is goals has been changed to emphasise the importance of the changes. First, whereas only simple products are sold in the current situation, salesmen needed packaged products to better fulfil the

customers' expectations. This is identified in requirement RPI1-2: a new goal "offer product catalogue" is adopted and replaces the former "construct a product list" goal. The refining of this gap includes changes in the salesmen activities, on the system functions, and in the way products are structured. Similarly, there is a requirement that the "manage booking contracts" goal becomes "manage customer relationship". It is also strongly required the customer relationship were maintained as long as possible, i.e. until exclusion of the customer. As requirements COS1-1, COS1-2, AS-4, AS-7, or AI-1 show it, this has numerous consequences, such as the de-multiplication of the strategies to manage the customer relationship. Applying the gaps of table 1 on the As-Is model results in the To-Be map shown in Figure 11. Of course this model is an incomplete view of the future system and business processes. It can be completed in several ways such as by goal refinement or by using complementary models. These are not shown here for the sake of space [4].

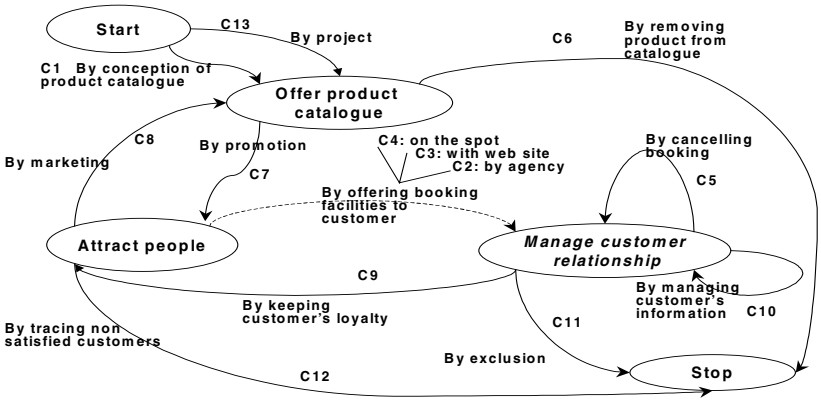

Fig. 8. The To-Be map

Achieving the required changes raises a number of instance-level questions such as: "How should we deal with unfinished contracts ?", "Can the new product catalogue be built from the existing flat product list?", "What should be done with the contracts made by third-parties while the new system is introduced?", "Can the new customer-centric IS be initiated with information gathered from recent contracts?", etc. These issues can be treated by analysing how the new business processes and system should comply with running As-Is instances. As shown in the following list, the outcome is a number of compliance gaps that specify the requirements resulting from the decisions that are taken :

- The structure of products in the old list and in the new catalogue are radically different. However, salesmen expect that customers and third parties keep referring to the old product list for a while before the transition to the new catalogue is completed. Besides, a detailed analysis shows that despite the differences, the initialisation of the new product catalogue from the old list could be automated. A compensation policy is thus adopted and procedures defined. At the requirement level, this decision is specified with a new gap according to which a strategy should be added to "offer product catalogue". This strategy is named "by reuse of the old one". The corresponding requirements indicate systematic equivalences between the two product structures.

- Besides, it is decided to let the product designers decide if they want to fully flush or abort the undertaken product design processes. In the former case, the "reuse" strategy can be used to include the created product in the new structure. In the latter case, the new catalogue construction strategies can be used to create aborted products in the new catalogue structure.
- The hotel owners asked for a smooth transition from contract-based to customer-centric room booking management. The idea is in fact to let these processes terminate the way they started, i.e. fully flush. Therefore, the new system should at least temporarily provide all the contract management services except for contract creation, as in the As-Is system, service.
- In addition to that, third parties asked to propose customers to improve their booking when these are based on a product that becomes part of a package. This can indeed be achieved using the until abort policy. The idea is to undo contract establishment processes up to the 'agreed' state, then implicitly migrate the contract proposals and start again with the To-Be strategy. Automated facilities are of course needed to support this. Compensation procedures are also required, for instance to re-fund the undone contracts that have already been paid.
- In so far as the termination of the overall sales process is concerned, a one-step flush is needed. The requirement is indeed that normal terminations should proceed As-Is whereas new procedures should be used for cancellations. The needed procedures should also keep track of the customer dissatisfaction as required by the C12 strategy which is added in the To-Be model.
- Contracts persist in the system for 5 years after they are terminated; therefore customer information is already available. All the stakeholders agreed that this information should be used to initialise the customer base of the new system. This requirement is specified by adding a new strategy "by using old customers' information" to achieve the "manage customer" goal. This gap is implemented by a number of complex compensation procedures which ultimate aim is to gather enough information on old customer to attract them towards the new products of the catalogue.

As shown in this case study, identifying business gaps is definitely not enough to grasp the customers requirements with respect to change. An additional step forward can be made to improve the new business and system by reasoning on the instance level. This leads to new requirements specifications that help ensuring the transition from As-Is to To-Be and further exploiting the existing situation to improve the quality of the To-Be models.

4 Related Works

The literature on change management addresses two different and complementary concerns: impact analysis and change propagation [16]. *Impact analysis* consists in predicting the impact of a change request before carrying out the software evolution. *Change propagation* involves the interpretation, in terms of gaps, of the consequences that changes have on a collection of models to maintain consistency. Both activities can be realised at different levels and thus be business-driven, model-driven or instance-driven.

Change management is *business-driven* when it is the study of businesses and of their context that drive impact analysis or change propagation. The hypothesis is that system evolution must fit business evolution which at its turn matches external forces like new laws, market evolutions, or the development of new technologies. As a result, understanding the business context evolutions helps understanding the future business processes which at their turn help better eliciting the requirements for the future system and thus guide its design. This business-driven change impact prediction approach is the one taken in [3] and [13].

In the case of *model-driven* change management, the idea is to use quality criteria that govern model development, to define change as a movement from a quality model to another quality model. For example, a number of goal map invariants are defined in [3] to complete gap models with new gaps to ensure the consistency of goal maps and bring the system in a new coherent state. Similarly, rules are proposed in [16], [17] or [18] to propagate gaps using invariants defined (and required) on models.

Strategies are proposed in [5], [6] and [7] to study the impact of a given change on the current instances of a model. For example, [5] proposes to use compliance graphs to migrate As-Is instances to the new model. Such approaches are interesting on the technical level, but provide no insight on how to collect the corresponding requirements. The same issue is raised with [6] and [7] which only propose to analyse the impact of change on the instance level.

5 Conclusion

It is now widely known that a very large part of IT development costs are due to maintenance. In the case of IS, part of the maintenance stands in fact in changes required to preserve the fitness of the system to the business that uses it. Our research program started by proposing a methodological framework for the engineering of system change in the context of enterprise evolution [13]. This framework was followed by a method to guide the elicitation of business requirements. This papers raises the question of the impact that such business requirements may have on the current situation. A number of policies adapted from the literature are proposed to reason on the modifications that should be made on As-Is models so that their running instances could be adapted to the future situation defined with To-Be models. In addition to an initial catalogue of modification policies, our proposal is to specify the modification requirements under the form of gaps called compliance gaps. There are thus business gaps that originate from business change requirements, and compliance gaps that originate from instance level modification policies.

Our research agenda combines research in two complementary : on the one hand, we want to further explore, and formalise the rules that govern gap identification using the compliance strategy. On the other hand, we believe that further work is needed to guide business gap identification. Domain knowledge reuse and case-based reasoning are two tracks that we are now exploring. Ultimately, our wish is to build a repository of exactable rules that could be used as a methodological encyclopaedia as well as to guide stakeholders pro-actively in their analysis of change.

References

1. Salinesi, C., Rolland, C.: F*itting Business Models to Systems Functionality Exploring the Fitness Relationship*. Proceedings of CAiSE'03, Velden, Austria, 16–20 June, 2003.
2. Salinesi, C., Presso, M. J. : *A Method to Analyse Changes in the Realisation of Business Intentions and Strategies for Information System Adaptation.* Proceedings of EDOC'02, Lausanne, Switzerland, September, 2002.
3. Rolland, C., Salinesi, C., Etien, A.: *Eliciting Gaps in Requirements Change*. To appear in Requirement Engineering Journal. 2003
4. Etien, A., Salinesi, C.: *Towards a Systematic Definition of Requirements for Software Evolution: A Case-study Driven Investigation*. Proc of EMMSAD'03 Velden, Austria, 2003.
5. Sadiq, S.: *Handling Dynamic Schema Change in Process Models*. Australian Database Conference, Canberra, Australia. Jan 27–Feb 02, 2000.
6. Liu, C., Orlowska, M., H. Li.: *Automating Handover in Dynamic Workflow Environments*. Proceedings of 10th CAiSE, Pisa, Italy, 1998.
7. Bandinelli, S., Fuggetta, A., Ghezzi, C.: *Software Process Model Evolution in the SPADE Environment*. IEEE Transactions on Software Engineering, 19(12) pp.1128–1144, (1993).
8. Jarke, M., Pohl, K.: *Requirements Engineering in 2001: Managing a Changing Reality*. IEEE Software Engineering Journal, pp. 257–266. November 1994.
9. Van der Aalst, W.: *Generic Workflow Models: How to Handle Dynamic Change and Capture Management Information*. In M. Lenzerini and U. Dayal, editors, Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems, pp. 115–126, Edinburgh, Scotland. September 1999.
10. Conradi R., Fernström, C., Fuggetta A.: *A Conceptual Framework for Evolving Software Process*. *ACM SIGSOFT Software Engineering Notes*, 18(4): 26–34, October 1993.
11. S. Sadiq and M. Orlowska. Architectural Considerations in Systems Supporting Dynamic Workflow Modification. Proceedings of the workshop on Software Architectures for Business Process Management at CAiSE'99, Heidelberg, Germany. June14–18, 1999.
12. Joeris, G., Herzog, O.: Managing Evolving Workflow Specifications With Schema Versioning and Migration Rules. TZI Technical Report 15, University of Bremen, 1999
13. Salinesi, C., Wäyrynen J.: A Methodological Framework for Understanding IS Adaptation through Enterprise Change. In Proceedings of OOIS'02, 8th International Conference on Object-Oriented Information Systems, Montpellier, France, September 2002
14. Rolland, C., Prakash, N.: Matching ERP System Functionality to Customer Requirements. In: Proceedings of RE'01, Toronto, Canada (2001), 66–75.
15. Rolland, C., Prakash, N., Benjamen, A:.A Multi-Model View of process Modelling, Requirements Engineering Journal, (1999) 4 : 169–187.
16. Han, J.: Supporting Impact Analysis and Change Propagation in Software Engineering Environments. In Proceedings of 8th International Workshop on Software Technology and Engineering Practice (STEP'97/CASE'97), London, UK, July 1997, pages 172–182.
17. Deruelle, L., Bouneffa, M., Goncalves, G, Nicolas, J. C.: Local and Federated Database Schemas Evolution An Impact Propagation Model. In Proceedings DEXA'99, pages 902–911, Florence, Italy, Aug.30–Sep. 4, 1999
18. Chauman, M. A., Kabaili, H., Keller, R. K., Lustman, F.: A Change Impact Model for Changeability Assessment in Object Oriented Software Systems. In Proceedings of the Third European Conference on Software Maintenance and Reengineering. IEEE Comput. Soc, Los Alamitos, CA, USA, 1999.