Managing Requirements in a Co-evolution Context

Anne Etien, Camille Salinesi
Centre de Recherche en Informatique, Université Paris 1 – Panthéon Sorbonne
90, rue de Tolbiac – 75013 Paris – France
aetien@univ-paris1.fr, camille@univ-paris1.fr

Abstract

Complex artefacts, such as Information Systems (IS), have multiple aspects and components: business processes, databases, architecture, or software. It is generally agreed that all these should be kept consistent over time. One major issue to preserve consistency is when required evolutions affect multiple aspects or components of the system at the same time. As each evolution requirement can have an impact onto several projects, teams, engineering domains, viewpoints, or system components, the question of "is the consistency link preserved by this requirement?" has to be continuously raised.

This paper presents: (i) a framework that defines challenges for RE caused by co-evolution and (ii) an approach to solve some of these RE-related co-evolution challenges. The framework was developed based on our experience in three IS evolution projects: ERP installation, baselining of an IS across subsidiaries, and business process improvement driven IS evolution. Each challenge identified in the framework is discussed with respect to our experience with practice and state of the art methods. Our approach was developed for the business process improvement driven IS evolution project, then generalised for the IS baselining project. The approach is presented, and then illustrated with the case of the latter project.

1. Introduction

Modifying a system can have multiple consequences. On the one hand, the initial system modifications can engender other ones, which are discovered through impact analysis [1]. On the other hand, once the system has evolved, a conceptual mismatch [2] can appear between the system and another entity such as its architectural environment or business processes, hence reducing the performance of the organisation [3]. This conceptual mismatch requires adapting the other entity to re-establish the alignment [4]. Such successive evolutions can obviously occur the other way round, which again causes system evolution.

One purpose of biology is among others to study interacting species, i.e. species that influence each other's evolution; this is called co-evolution. Similarly researches have been achieved in computer sciences to analyse the reciprocal evolution of systems or software and other entities such as organisations [5], business processes [4], or environment [6]. This paper proposes to take a look at co-evolution on a requirements engineering perspective.

Requirements engineering can be seen as a way to establish a relationship between the "why" and the "what" of the system under development [7], [8]. The latter deals with the system functionality whereas the former provides its rationale. A global view on co-evolution can therefore be taken by analysing the rationale for making systems or software evolve in coordination with other entities.

This paper first proposes a framework to define the qualities expected from RE approaches that aim at *understanding co-evolution* and *engineering it*. The framework is structured around five dimensions. Each dimension corresponds to a RE-related issue that we see as a key to understanding and engineering co-evolution. Different approaches can be taken to solve each issue. These approaches are presented as values in the dimension that corresponds to the issue. RE methods are positioned in the framework by identifying which value they take in each of the five dimensions.

The next section presents the dimensions of our framework that deal with the issues of *understanding* co-evolution. Section 3 presents the dimensions that deal with the issues of *engineering* it. Section 4 presents our approach to deal with co-evolution though an application example inspired from an industrial collaboration with the financial branch of the French

Proceedings of the 2005 13th IEEE International Conference on Requirements Engineering (RE'05)
0-7695-2425-7/05 $20.00 © 2005 IEEE

car constructor Renault. The concluding section uses the framework to discuss our position with respect to state of the art and emphasises open issues of co-evolution in RE.

2. Understanding Co-evolution

One issue when a system evolution project is undertaken is to handle the consistency of the evolving system with other co-evolving entities. Another issue is to express the requirements which specifically relate to system evolution in an adequate way. These issues are respectively tackled by the framework in the dimensions named "understanding relationship" and "expressing evolution requirements". Each dimension is presented in the next two sub-sections.

2.1. Dimension 1: Understanding relationship

Handling the consistency of an evolving system with other co-evolving entities requires understanding the consistency relationship between those. At the requirements level, the presence of different entities can introduce difficulties to understand the consistency relationship. These difficulties are caused by the use of heterogeneous languages, the fact that documentations are physically separated, or because the entities have different system-subsystem decompositions.

For example, when a business and an information system co-evolve, different languages are used to deal with the business level, and with the system level. Business models use concepts such as goals, processes, actors and roles whereas system functionality models deal with objects, operations, events and the like. Understanding if these entities interact and cooperate properly, they need to be linked by a consistency relationship. Such a link allows to know how the two entities are aligned in the current state, and after co-evolution. It is also useful to decide on when to undertake co-evolution.

Different approaches exist to define such relationship: traceability, links/rule typologies, metrics, and common languages. Each produces a value in the "understanding relationship" dimension.

Traceability "makes it possible to look at a change to a requirement and to find those parts of the design and code details that are affected by the change" [9]. Pohl also uses traceability to understand the relationship between different products of the same system used in different phases of an evolution project [10].

Typologies of links or rules help to define in a formal way what consistency between models expressed with different languages is. For example, Landtsheer and al [11] propose a technique for deriving event-based specifications, written in the SCR tabular language, from operational specifications built according to the KAOS goal-driven requirements language. [12] identifies links between concepts of the i* meta-model and those of the Z language. The approach specifies how changes on an i* model can be translated into modifications of Z specifications.

Authors like [13] propose to define consistency using *metrics*. The other way round, [14] suggests that identification of unfit requires the application of a fit measurement method.

Using a *common language* is useful to materialize the alignment between two entities. This approach is used by Clarke for which "to address the misalignment of design and code, one approach is to impose the same development paradigm on all software artefacts – both are written in the object oriented paradigm" [9]. This idea is also conveyed by SysML [15], which purpose is to adapt UML to multiple disciplines involved in systems engineering projects, such as electronics or mechanics, so as to understand how the different models used in these disciplines integrate with each other.

Our experience in information system evolution projects showed us that there is a great need in the industry to document the level of alignment of information system components with each other and with business models. Our approach is to express alignment using a common language, namely oal/Strategy Map, [4].

A map is represented with a directed graph in which nodes are labelled with goals and edges labelled with strategies. Having several edges pointing to the same node allows to represent the different strategies available to achieve the same goal. The directed nature of the graph is a way to represent the flows of goals. Therefore, a *map* can be defined as composed of several sections where each section is an aggregation of two kinds of goals, *source* and *target*, linked together with a *strategy*. see Figure 6 or [16] for more details .

The coupling between business processes and system functionalities is achieved in the map formalism by simply relating map sections to the business processes and system functionalities that they abstract. Therefore, a map materializes the alignment between business processes and system functionalities when both can be abstracted into the same goals and strategies. We believe that the oal/Strategy Map language is adequate to understand the consistency relationship because it is intentional by nature, and thus can be interpreted both in the perspective of the business and in the perspective of the system. Consistency issues are raised when there is a system

goal that does not match a business goal, or the other way round when a business goal cannot be achieved using the system.

2.2 Dimension 2: Expressing evolution requirements

Requirements must be expressed in evolution projects as in any other system development projects. In some cases, the requirements are expressed as if the system was developed for the first time. In other cases, the requirements document is produced from an older one and takes into account the evolutions without making them really explicit. In both cases, the requirements documents only deals with the new system. In a third kind of approach, the gaps and similarities required between the new system and the older one are made explicit under the form of evolution requirements specifications. Specifying the evolution requirements is a concern that differs from making the requirements documents evolve. The three values in the "expressing requirements" dimension are: 1 from scratch, 2 by requirements evolution, and 3 through evolution requirements.

In many projects, requirements are expressed as if *from scratch*. Several approaches as Merise, i*, Kaos or the RUP do not provide indications concerning evolution of the current models. They help to construct a new one. As Figure 1 shows, the focus is on the future To-Be situation. The current situation can be ignored either because it is too poorly documented, or because the future one is too different from it. Poor documentation of the legacy is a situation frequently met in organisations that have developed their information system a long time ago and where many evolutions have occurred without concern for overall integration. ERP installations are typically projects in which the future system and business processes are likely to be so radically different from the current ones that comparing those would be too costly or simply meaningless.

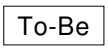

Figure 1. Expressing requirements from scratch: focus on the future

In practice, requirements management tools such as RequisitePro or Doors are often used to make *evolve* the requirements documentation when evolutions are required. At the requirements level, an evolution results in adding, removing or changing a requirement in the requirements documents. The transition from a new situation As-Is to the new one To-Be shown in Figure 2 occurs when the requirements documents is released. In this approach, the required evolutions are kept implicit. Therefore, a retrospective analysis must be achieved to understand what will actually evolve and to conduct change itself. However, the advantage of this approach is that requirements that do not change do not have to be specified again. Besides, approaches such as [17] [18] are able to exploit successive modifications of the requirements documents to keep the specification of the future system permanently up to date and consistent.

Figure 2. Expressing requirements through requirements evolution: the evolution required is implicit

Evolution requirements can be made explicit just as any other requirement can be. As shown in Figure 3, evolution requirements are in-between the As-Is and To-Be situations: they express the transition required between the old system and the new one. Languages to express evolution requirements focus on the gaps and on the similarities between the old system and the new one.

Evolution requirements are typically defined under the form of gaps when the evolution is limited and a large part of the system shall remain as is.

Specifying evolution requirements under the form of gaps allows to identify the impacted part of the existing system, and to automatically check the consistency between the requirements for the future system and the required evolutions.

Different gap and similarity based languages have been defined in different domains to specify evolution requirements. [19] proposes similarities formulae to reuse source code, [20] uses other formulae in an object oriented model context whereas [21] compares UML models. [22], [23], and [24], allow to express requirements of database schema evolutions, the language proposed by [25] can be used to expressed evolutions required on XML DTDs, requirements of evolution of workflow model can be specified using languages such as the ones defined by [26], [27], or [28]. Each language is defined under the form of a collection of operators that specify a transition that can be typically required between models of the As-Is and To-Be situations. The main advantage of this approach is that it allows to express evolution requirements in a synthetic way. Besides, it makes explicit which part of the current system is impacted by evolutions, and under what constraints it shall evolve. However, such approach can only be used when the As-Is and To-Be situations are defined with semi-formal or formal models.

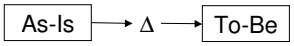

Figure 3. Expressing explicit evolution requirements as a transition between the current and the future

We proposed in [29] a generic requirements evolution language under the form of a typology of gap operators that can be specialised for any modelling language. Once specialised for the oal/Strategy map formalism, our language can be used to express how gaols and strategies shared by the business and the system shall change, e.g. a goal can change of name, a strategy can be replaced by another one, sections can be merged or the other round split, etc. Combined with maps of the current situation, evolution requirements expressed with this language can be used to automatically control the consistency of the requirements actually released for the future system under the form of goal/strategy maps.

As required by [27] and [24], our typology of operators is complete and consistent [30]. Completeness refers to the possibility of expressing any type of evolution requirement; correctness relates to the problem of expressing an evolution requirement that does not generate an issue in the future system i.e. by preserving a set of correctness invariants [24] . Besides, our approach is exhaustive as only one gap operator is used to express each evolution requirement [30]. Most of the evolution requirements languages are complete and consistent. However, these languages often provide the simplistic CRUD view on evolution, which results in difficulty to express complex evolution requirements [25].

Our approach can be adapted to other requirements modeling languages such as KAOS, I*, L'Ecritoire, or bram. Its drawback is that it cannot be used in projects in which requirements are not modelled, i.e. only specified in natural language.

The three values of the dimension "Expressing evolution requirements" are: from scratch, requirements evolution and evolution requirements.

3. Engineering Co-evolution

Co-evolution involves both impact analysis and change propagation. The purpose of *change propagation* is to carry out evolution requirements into the To-Be [1]. *Impact analysis* aims at evaluating how evolution requirements can affect the internal consistency of the evolving system and its compliance with other entities and identify modifications that are estimated necessary to preserve consistency.

Different ways of working can be used to engineer co-evolution. In any case, it is necessary to i identify estimated modifications from initial ones, ii carry out these evolutions and iii check the consistency of the evolutions of co-evolving entities. Each of these three aspects is treated in the following sub-sections.

3.1. Dimension 3: Eliciting evolution requirements

The question raised through the elicitation issue is that of impact analysis: how to elicit evolution requirements taking into account that several entities have to evolve at the same time. Figure 4 shows that impact analysis approaches can be divided into four families, namely independence, interdependence, dependence, and double dependence. Each family is defined according to the direction of the dependency relationships between the evolving entities.

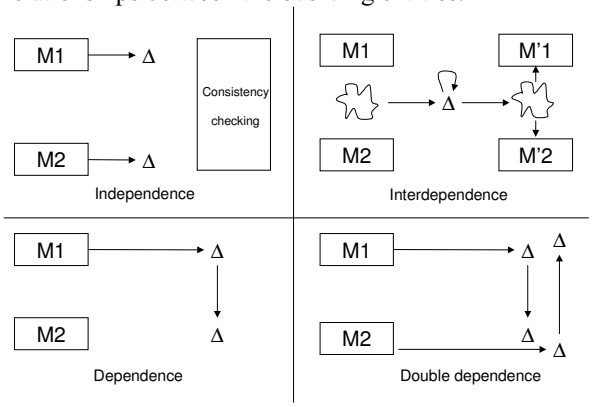

Figure 4. Dependence approaches to engineering evolution requirements of co-evolving entities

Co-evolution is engineered *independently* left upper corner of Figure 4 , when there is no dependency between the engineering processes of each evolving entity. This is typically the case in projects under high time pressure: evolution requirements are directly implemented into the system without checking consistency with the target business organisation. This approach is also taken for evolutions projects that involve systems that shall disappear in the short term, or when the evolution requirements are themselves highly volatile and likely to be quickly abandoned.

Independent co-evolution often goes along with projects in which evolution is implicit or when the number of co-evolving entities is too high. This situation is well illustrated by systems engineering projects involving many different engineering disciplines. Not only the evolution requirements are often implicit different requirements documents are even sometimes released at different moments , but also it often occurs that it is technically difficult to understand how an evolution required in one discipline

impacts the system in the perspective of the other disciplines.

To solve this issue, consistency can be checked in retrospect to verify that the new system is aligned with other entities [31]. Alignment can also be evaluated using metrics [13]. If misalignment is observed, other evolutions are needed and a new engineering cycle can be performed. This is typically what happens in ERP projects where the customisation of an individual transaction has to be implemented to check through non-regression tests that there is no impact on other transactions or modules.

In the *dependent* approach to eliciting evolution requirements, the evolution requirements of an entity are deduced from the evolution requirements elicited for another co-evolving entity. This is typically the kind of approach taken in a business process improvement project when information system evolution requirements are generated to comply with evolutions required at the business level. This approach is also frequently taken by managers of projects portfolio who define master projects with which other information system evolution projects should comply. This approach is formalised by MDA methods. At the requirements level, rules are for example proposed by [12] to deduce evolution requirements on system specifications expressed in Z from evolution requirements expressed at the intentional level with I* models. In a dependent approach, consistency is checked as an external property where the dependent requirements should comply with a dependee entity, e.g. domain knowledge in [18].

There is a *double-dependence* when each co-evolving entity can play the role of master in the impact analysis of evolution requirements. For example, [32] proposes rules to evaluate the impact of evolution requirements specified at the business level on system-level evolution requirements and vice-versa. A double dependence approach can be considered as the combination of two one-way dependence approaches.

On the contrary, an *interdependent* approach can be considered two-way. Indeed, in this kind of approach each evolution requirement specifies how all co-evolving entities evolve at the same time. Then, impact analysis and propagation are performed with a single collection of evolution requirements. This can be achieved using invariants [24], or using heuristic rules as suggested by [1].

Our approach uses a unique model that integrates a business process perspective and a system functionality perspective. It is thus interdependent. The approach gives the same importance to the two co-evolving entities since each evolution requirement is expressed for both of them. Our experience in industrial projects showed us that the risk lies in the propagation of the evolution requirements on the actual specifications of the co-evolving entities. Indeed, even when systematically guided by mapping rules, the double propagation can generate inconsistencies as soon as design decisions are required. We propose to handle this issue using goal-by-goal top-down exploration of our models to ensure that the impact of each design decision was immediately evaluated on both co-evolving entities business process and information system , and therefore that consistency was preserved [29].

This dimension has thus four values: Independence, Dependence, Double dependence and Interdependence.

3.2. Dimension 4: Propagating changes

The process of carrying out the initial and the estimated modifications is called *change propagation* [1]. Once evolution requirements have been elicited, specifications of the new system have to be produced. In other terms, this issue focuses on the way to design the To-Be models. We found that this part of the co-evolution process is seldom described in literature or documented in projects. Two families of approaches can however be defined: symmetric and asymmetric approaches.

The most current approach is illustrated in Figure 5 on the left hand side. It consists in combining the evolution requirements with the As-Is model of each evolving entity, so as to obtain the corresponding To-Be models [12]. This approach is called *symmetric propagation* because specific evolution requirements are symmetrically combined to each of the co-evolving entities [33].

In the *asymmetric propagation* approach, evolution requirements which were designed to generate the To-Be model of an entity is also used to generate the to-Be model of the other co-evolving entity [33].

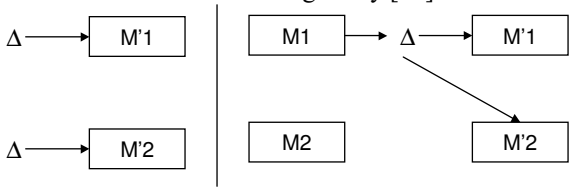

Figure 5. Symmetric and asymmetric evolution requirements propagation

Our approach consists in propagating evolution requirements expressed in reference to a goal model onto business process models and functional models of the system. At first sight, it is therefore twice asymmetric. Our propagation from evolution requirements to the target models is however not direct. We decided to first build a consistent intentional

model that integrates the view of each co-evolving entity the business and the system . Then, the intentional To-Be model is used to guide evolution requirements propagation on the business process models and on the functional models of the system. The initial symmetric propagation of evolution requirements onto the To-Be goal/strategy maps is thus useful to lower the drawbacks reported in section 3.1.

Two values symmetric and asymmetric have been identified for the change propagation dimension of the framework.

3.3. Dimension 5: Verifying relationship between the To-Be entities

"With no suitable policy, inconsistencies provoked by changes in one of the two concerned sets usually lead to irremediable erosion and the very common situations where, for example, architectural artefacts are no longer updated" [34]. Co-evolution aims to bring the To-Be models in a state where the consistency relationship holds between the different entities. Practice shows that drifts occur through time. Different approaches can be taken to check the consistency relationship and manage the drift.

The most current way to check consistency in practice is through integration tests. This approach has been widely documented and is well spread in practice. In some situations, such as the one illustrated in the upper left corner of Figure 4, it is very difficult to avoid it. However, its drawback is also very well known: the later an issue is identified, the more costly and risky it is. Other approaches are thus necessary to check consistency earlier; in particular at the requirements level.

Consistency can be checked through *dependence links*, each time an evolution requirement is elicited. In that case, each time a couple of evolution requirements is elicited, the two entities are consistent. In theory, it is the links between requirements that assures that the resulting models of the co-evolving entities are consistent. However, this approach does not provide a global view on the evolution requirements. Evolution requirements can be complex, and be in conflict with one another without being identified by the dependency links.

On the opposite a global view can be established on consistency using *metrics*. In this approach, the consistency relationship is established by consistency metrics that measure the "distance" between models of the co-evolving entities. A drift is identified when the metrics show that the evolution requirements that were elicited lead to a loss of consistency. In this case, either new evolution requirements should be elicited, or the newly elicited requirements should be revised. Adaptations are then made until full consistency is re-established [33]. A list of ten metrics is available in [35]. For example, the *support ratio*, similar to the criterion defined by Bodhuin [13] is the extent to which business activities are supported by the system. This metric is calculated as follows:

$$\text{Support ratio } Sr = \frac{\text{Number of activities represented by system events}}{\text{Number of activities}}$$

The higher this ratio is, the more automated the activities are. Conversely, a low support ratio expresses that a large number of business activities are manually carried out and implies that new evolution requirements have to be elicited to better align the business and the system. However, a high value for the support ratio does not signify that business and system are completely aligned. For this reason we define other metrics.

Obviously, to avoid a misfit between the business and the system there should be a strong correspondence between the business information and the system information. This can be measured by the fact that there exist things in the system that *map* business things. The *informational completeness criterion* measures the proportion of this mapping and can be written as follows a business object being for example, a client, an account, or a demand :

$$\text{Informational completeness } Ic = \frac{\text{Number of business object mapping system object}}{\text{Number of business object}}$$

These two metrics as the eight other rely on generic metrics, which have been defined using i two generic models to represent the business and system and ii links that have been identified between concepts of these two models. The generic metrics are adapted to specific business and specific system models.

4. Application example

This section illustrates our co-evolution approach using an application example inspired from an industrial collaboration, in which stakeholders wanted to make evolve both their business processes and the functionalities of their information system. The first sub section, gives an overview of the project. Then, the As-Is business processes and system functionalities are presented using the goal/strategy Map formalism. Then the application of our co-evolution requirements elicitation approach is illustrated.

4.1. Overview of the project

DIAC is a subsidiary of the financial branch of the French car constructor Renault. Besides providing

other financial services, DIAC's information system deals with granting credit to French Renault customers. After several mergers and take-overs, a number of similar systems are running at other subsidiaries located in different countries. It was required to standardise these across Europe by adapting the Spanish software system in France, Spain, Portugal, and ermany.

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 too: a adopting a client driven strategy and no more a product-oriented one, b including additional financial services, such as personal loans in addition to car loans and c planning for the same software system to be used across Europe.

The first concern of the project was to evaluate how to make co-evolve the Spanish system so that it complies with French business processes and the Spanish business processes so that it complies to new ways of working . As the current situation was poorly documented in so far as French business processes were concerned and in a foreign language in so far as the Spanish system is concerned, we proposed to start by developing a high-level view of the current situation with intentional models, rather than a time-taking and costly translation of detailed specifications and detailed audit of all French business processes. The language selected to specify the As-Is situation was that of goal/strategy Map. A complete documentation was produced in about a month. The document is 120 pages long and describes 34 models.

4.2. Presentation of the As-Is situation

The overall objective of the company is to sell financing products credits and leases associated to vehicles manufactured by the Renault group. DIAC's main business goal is traditionally split into on pre-sales and post-sales. Pre-sales involves building product catalogues, marketing, and making contracts with customers. Post-sales includes treasury, coordination with partners, customer management, and information flow management. For the sake of space, this paper only deals with post-sales, and more precisely with repayment of loans.

Each contract defines a schedule of repayment, which determines the amount of each monthly recovery. The repayments received by DIAC must be allocated to contracts and the schedules are monitored to check late or incomplete payments. The repayment item can correspond to 1 the contractual schedules represented in Figure 6 by the section <*Manage schedule repayment item*, *Manage schedule repayment item*, *by payment*> or 2 the schedule resulting from renegotiation after unpaid debts materialised in Figure 6 by the strategy *by no payment* .

It is also necessary to *terminate the contract*. A contract can be closed *normally*, i.e. after total loan repayment. A contract can also be terminated *by anticipation*; in that case, for example, the client asks for paying several repayment items at the same time. The *by contract revocation* strategy corresponds to a contract cancellation at the client initiative in the legal revocation period. In every case, ending a contract engenders different activities: to *record administrative operations*; to manage the residual debt either *by friendly agreement* or *by bone of contention*. The strategy *by archiving* that ends the process corresponds to a legal obligation to materially keep contracts during several years.

This map also indicates how the system works. For example, the strategies *by invoicing* and *by manual archiving* respectively imply that i an invoicing module and ii a contracts editing and printing module exist in the system. When *no payment* is received for an item, the system triggers a procedure to identify the unpaid debts and define a new repayment schedule, or abandon the debts, or even begin a contentious procedure. Similarly, the *by anticipation* strategy introduces some flexibility in the system in the sense that it allows to cash in several repayment items at a time.

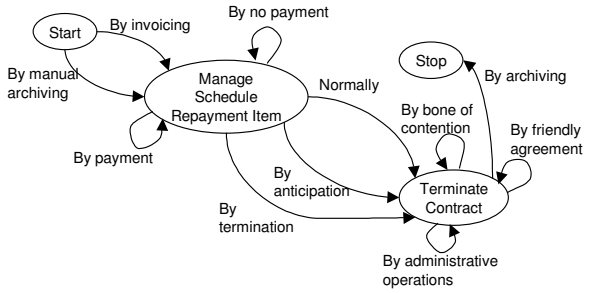

Figure 6: *Manage the customer relationship by loan recovery*

The map presented in Figure 6 corresponds to the Spanish system and business processes before standardisation across the French subsidiary.

It was chosen to express evolution requirements under the form of gaps between the current and the future goals and strategies. A set of evolution gap operators was specifically developed to match with the Map formalism. The proposed evolution requirements language included gap operators such as merge sections, split goals, change the target of a strategy, rename a strategy, add a strategy, and the like. All the operators defined in the resulting evolution language cannot be reported here for the sake of place, but a

more complete description can be found in [29]. Most of the operators that we proposed were used in the projects to express evolution requirements.

4.3. Engineering co-evolution

It was chosen to represent the relationship between the two co-evolving entities with a unique model, the Map formalism. Evolution requirements were thus elicited by analysing interdependencies between the two entities business processes and information system functionalities .

Table 1 gathers some of the evolution requirements that were initially identified based on the map *Manage the customer relationship by loan recovery*. The table shows for example, that business processes and system functionality implementing the section *<Start, Establish customer relationship, by welcome>* had to be added so as to improve customer relationship. The purpose of this section is to welcome and to communicate on essential information of the contract. This evolution requirement results both i in business process modifications since mail and/or phone have to be given to each client and ii in system evolutions since information concerning the client must be available from a secure area on the Internet.

The Spanish system does not allow direct payment by bank transfer on the order of the customer, because an invoice must be sent to the client before the company receives the payment. The gap corresponding to change the origin of the strategies *by invoicing* and *by payment* allows to dissociate these two activities and eventually to manage them at the same time in the future. The business processes are thus modified i to allocate payment directly to the client account and ii to henceforth accept direct debit payment. In the system, the implementation of the business rules have to change; invoice i is no more considered as a precondition to the payment and ii it should be possible to conserve the customer banking information, thus modifying the structure of the customer class if the system is modelled with a UML class diagram.

Contracts have to be physically archived by an external provider. In the system to be, it should be possible to numerically archive the contracts via an electronic management of the documents. This requirement is materialised by replacing the strategy *By manual archiving* by *By contract archiving*. It well corresponds to a replacement insofar as the activity corresponding to this section has changed. It is thus not only a naming problem. This gap introduces modifications in the system as adding the module to electronically manage documents. This implies the construction of a new database or an adjunction of a new part to the existing one.

One of the new business needs is to include additional financial services. This, on the one hand, changes the conceptual structure of a contract. On the other hand, this implies to contact the services providers to transfer money and inform them in case of anticipated end of contract respectively corresponding to the gaps 'adding *By transferring money to service providers*' and 'adding *By information to service providers*' . This implies to add new actors in the business model. Finally, it is necessary to distinct the end of the repayment and the end of the contract. The intention *Terminate the contract* is thus split into *Terminate the contract management* and *Terminate the repayment schedule*. Thus, the complete loan repayment does not imply that the additional services end. In that case, it is important to differentiate the repayment and the contract terminations. Such evolution requirements imply modifications in the system. Indeed, the customer relationship and the contract have eventually to be maintained after the complete loan repayment until the end of the additional services. Client must be able to terminate, by anticipation, his/her repayment of schedule and/or his/her contract. This has for consequences adding the strategy *by withdrawal* to end a contract and changing the origin of the strategy *by anticipation*.

Operator	Element
AddStrategy	*By withdrawal* *By transferring money to service providers* *By information to service providers*
AddSection	*<Start, Establish the client relationship, by welcome>*
ReplaceStrategy	*By manual archiving* by *By contract archiving*
SplitIntention	*Terminate the contract* into *Terminate the contract management* and *Terminate the repayment schedule*
ChangeOrigin	*By invoicing* source intention to *Establish the client relationship* *By payment* source intention to *Establish the client relationship* *By anticipation* target intention *Terminate the repayment schedule*

Table 1. Gaps between the As-Is and To-Be maps.

For sake of place, only few evolution requirements have been presented in Table 1 corresponding to the gaps identified above. For each evolution requirement, we tried to show how an interdependence approach helps us to elicit new requirements either on the business or on the system. During the project, only for this map, 21 gaps have been identified all having an impact on the system and the business. lobally, around 350 evolution requirements have been elicited. Sometimes these gaps have been identified since the construction of the As-Is models by the French or Spanish business specialists. Indeed, we worked by group of three persons, a French and a Spanish

business specialist and one of us to construct the models.

Once the To-Be maps have been constructed, it was necessary to design both the future system and the future business processes using respectively, on the one hand a class and an event UML diagrams and, on the other hand, a use case and an activity UML diagrams. Finally, use of metrics helped us determining if the business and the system were aligned. For example, the support ratio and the information completeness metric respectively indicating the rate of the supported activity and the rate of the business objects supported by the system were equal 1 and 0.97 1 being the maximum . This meant that each activity existing in the system could be performed with the system and that almost each business object corresponded to a system object. The two To-Be entities were not completely aligned but the result was satisfying insofar as before the project, the metrics only reached 0.6 in average. It was possible to improve the consistency relationship between the business system by introducing new object in the system or removing from the business some others that did not map with a system object.

5. Conclusion

In this paper, a framework presenting five dimensions of co-evolution management has been proposed. Each dimension represents an issue. Different ways to handle each issue have been drawn from the literature thus establishing values in the framework.

Our framework also situates our approach comparatively to other one as shown in Table 2. The black squares specify that this point is clearly explained by the authors whereas the grey squares result from deduction reading descriptions of a given approach.

Dimension	Values	Clarke	Krishna	Bodhuin	Zowghi	Mens	Kardasis	Our approach
Understanding Co-evolution	Traceability	■						
	Links or rules		■		□	□		
	Metrics			■				
	Common language							■
Expressing evolution requirements	From scratch	□		□				
	Requirements evolution				■	□		
	Evolution requirements						□	■
Eliciting evolution requirements	Independence					■		
	Dependence	□	■		□			
	Dependence double						■	
	Interdependence							■
Propagating Changes	Symmetric	□	■		□	□		
	Asymmetric							■
Verifying relationship	Requirements driven							
	Model driven		■					
	Using Metrics			□				■

Table 2. Positioning different approaches with the framework

Table 2 highlights that 1 there is not a single way to deal with co-evolution; advantages and disadvantages were discussed in the framework description. We believe that a situational method integration could help define and integrate approach able to deal with different co-evolution situations. 2 some aspects of co-evolution are not very well treated. The reason can be threefold: our review of the state of the art is not complete, the issue is not as important as we thought, the issue is important and not well treated, therefore it should be treated in future research. Our experience in several projects with the industry showed us that this tend to be the case.

Our approach has the advantage i to use the map formalism as a common language to represent both the system functionalities and the business processes; and ii to be based on explicit evolution requirements. The first point allows to gain time when the As-Is models are not up to date and to use a language understandable by everyone, the stakeholders as well the technicians or the users. The second point allows to focus only on what changes. Our approach seems the only one to propagate changes symmetrically or to elicit evolution requirements through interdependence. However, this results from the choice to represent the relationship between two entities with a common language. This approach has been evaluated on several projects and continually evolves to answer new issues met in industrial context.

Our agenda relies on two key axes: i a definition of eventually generic rules or guidelines to pass from the Map formalism to the two meta-models ii an extension of our approach to allow co-evolution between different entities and not only between system functionalities and business processes.

6. References

[1] J. Han, "Supporting Impact Analysis and Change Propagation in Software Engineering Environments", *Proceedings of Workshop on Software Technology and Engineering Practice* STEP'97 / CASE'97 , IEEE Computer Society Press, London, UK, July 1997, pages 172-182.

[2] A. Arsanjani, J. Alpigini, "Using rammar-oriented Object Design to Seamlessly Map Business Models to Component-based Software Architectures*", Proceedings of the International Symposium of Modelling and Simulation*, Pittsburgh, PA, USA, 2001, pp 186-191.

[3] Henderson, J. C. and N. Venkatraman 1993 . "Strategic Alignment: Leveraging Information Technology for Transforming Organizations". *IBM Systems Journal*, 32 1 : 4-16.

[4] C. Salinesi and C. Rolland, "Fitting Business Models to Systems Functionality Exploring the Fitness Relationship". *Proceedings of CAiSE'03*, Velden, Austria, 2003.
[5] M.M Lehman, J.F Ramil and Kahen, "Evolution as a Noun and Evolution as a Verb", *Proceedings of Workshop on Software and Organisation Co-evolution*, London, 2000
[6] E. Mitleton-Kelly and M-C Papaefthimiou "Co-evolution & an enabling infrastructure: a solution to legacy?", Systems engineering for business process change', Peter Henderson, Springer-Verlag, 2000, ch. 14
[7] A. van Lamsweerde. " oal-Oriented Requirements Engineering: A uided Tour". *IEEE International Symposium on Requirements Engineering*, Toronto, August, 2001, pp. 249-263
[8] E. Yu. "Agent Orientation as a Modelling Paradigm". *Wirtschaftsinformatik*. 43 2 April 2001. pp. 123-132.
[9] S. Clarke, W. Harrison, H. Ossher and P. Tarr. "Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code", *Proceedings of Object-Oriented Programming, Systems, Languages and Applications* OOPSLA , 1999.
[10] K. Pohl, S. Jacobs, "Concurrent Engineering: Enabling Traceability and Mutual Understanding", *Journal of Concurrent Engineering Research and Application, Special Issue on Concurrent Engineering and Artificial Intelligence*, Vol.2, No.4, 1994, pp. 279-290
[11] R. de Landtsheer, E. Letier, A. van Lamsweerde, "Deriving Tabular Event-Based Specifications from oal-Oriented Requirements Models", *Proceedings of RE'03, IEEE International Conference on Requirements Engineering*, Montgomery Bay, USA, September 2003.
[12] A. Krishna, A.K. hose, S. Vilkomir, "Co-Evolution of Complementary Formal and Informal Requirements", *Proceedings of International Workshop on Principles of Software Evolution* IWPSE'04 , September, 2004, Kyoto, Japan, pp. 159-164
[13] T. Bodhuin, R. Esposito, C. Pacelli and M. Tortorella, "Impact Analysis for Supporting the Co-Evolution of Business Processes and Supporting Software Systems", *Proceedings of BPMDS'04*, Riga, Latvia, 2004.
[14] P. Soffer, "Fit Measurement: How to Distinguish Between Fit and Misfit", note for BPMDS'04, Riga, Latvia, 2004
[15] http://www.sysml.org
[16] C. Rolland and N. Prakash. "Matching ERP System Functionality to Customer Requirements", *International Symposium on Requirements Engineering (RE)*, Toronto, Canada, August 2001
[17] D. Zowghi, A.K. hose and P. Peppas, "A Framework for Reasoning about Requirements Evolution", *Proceedings of PRICAI96*, Australia, 1996.
[18] D. Zowghi and V. ervasi "On the Interplay Between Consistency, Completeness, and Correctness in Requirements Evolution," *Information and Software Technology*, 45, 14 November 2003 , pp. 993-1009
[19] L. Prechelt, . Mahpohl, and M. Phlippsen. Jplag: *Finding Plagiarisms among a set of Programs*. Technical Report 2000-1, Universitat Karlsruhe, March 2000.
[20] A. Sarireta and J. Vaucher. "Similarity Measure in the Object Model". *Proceedings of. ECOOP.97*, Jyvaskyala, Finland, 1997.
[21] M.C. Blok and J.L. Cybulski. "Reusing UML Specifications in a Constrained Application Domain". *Proceedings 5th Asia Pacific Software Engineering Conference* pp. 196-202. Dec. 2-4, 1998.
[22] C. Delgado, J. Samos and M. Torres. "Primitive Operations for Schema Evolution in ODM Databases" *Proceedings of Object Oriented Information Systems*, Springer-Verlag Lecture Notes in Computer Science, Vol. 2817, eneva, Switzerland, 2003, pp. 226-237.
[23] S. E. Lauteman. „Schema Versions in Object-Oriented Database Systems", *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications*, Melbourne, Australia, April, 1997
[24] J. Banerjee, W. Kim, H.J. Kim and H.F. Korth, "Semantics and Implementation of Schema Evolution in OODB", *Proceedings of ACM SIGMOD Conference on Management of Data*, ACM, 1987, pp. 311-322.
[25] L. Al-Jadir, "Once Upon a Time a DTD Evolved into Another DTD", *Proceedings of Object Oriented Information Systems*, Springer-Verlag Lecture Notes in Computer Science, Vol. 2817, eneva, Switzerland, 2003, pp.3-17.
[26] M. Kradolfer. "A Workflow Metamodel Supporting Dynamic, Reuse-based Model Evolution". PhD thesis, Department of Information Technology, University of Zurich, Switzerland, 2000
[27] F. Casati, S. Ceri, B. Pernici and . Pozzi, "Workflow Evolution", *Proceedings of the Conference On Conceptual Modeling* ER'96 , Cottbus, ermany, 1996, pp. 438-455.
[28] M. Reichert, S. Rinderle and P. Dadam, *A Formal Framework For Workflow Type And Instance Changes Under Correctness Constraints*, UIB-2003-01, April 2003
[29] C. Rolland, C. Salinesi, and A. Etien, "Eliciting aps in Requirements Change", *Requirement Engineering Journal*, Vol. 9, 2004, pp1-15
[30] A. Etien and C. Salinesi, "Towards a Systematic Definition of Requirements for Software Evolution: A Case-study Driven Investigation". Proceedings of EMMSAD'03, Austria, 2003.
[31] T. Mens, A. Eden "On the evolution complexity of design patterns" *Proceedings of Software Evolution through Transformations SETra*, Rome, Italy, October, 2004
[32] Kardasis, P. and Loucopoulos, P. "Aligning Legacy Information Systems to Business Processes", Proceedings of CAiSE*98, Pisa, Italy, 1998, pp. 25 - 40
[33] R. Lämmel "Coupled Software Transformations", *Proceedings of International Workshop on Software Evolution Transformation* SET2004, 2004, pp31-35
[34] J.M. Favre, "Meta-Model and Model Co-evolution within the 3D Software Space" *workshop on Evolution of Large-scale Industrial Software Applications* ELISA03, Amsterdam, Netherlands, September 2003
[35] Etien A. and Rolland C. "Measuring the Fitness relationship", submitted to the special issue Coordinated Development of Business Processes and their Support Systems of the *Requirements Engineering Journal*, 2005