A Systematic Approach to Express IS Evolution Requirements Using Gap Modelling and Similarity Modelling Techniques

Camille Salinesi, Anne Etien, and Iyad Zoukar

CRI - Université Paris 1, 90, rue de Tolbiac, 75013 Paris, France
camille@univ-paris1.fr
{anne.etien,iyad.zoukar}@malix.univ-paris1.fr

Abstract. Gaps and similarities are two important concepts used in Information System (IS) projects that deal with the evolution issue. The idea in using these concepts is to analyse what changes or what remains similar between two situations, typically the changed situation and the new one, rather than just describing the new situation. Although in the industry, the daily practice consists in expressing evolution requirements with gaps and similarities, little attention has been paid in research to better systematically define these two kinds of concepts so as to better support the expression of evolution requirements. This paper proposes an approach that combines meta-modelling with generic typologies of gap operators and similarity predicates. Our purpose is not to define yet another requirement modelling language. On the contrary, the two generic typologies can be adapted to existing modelling language such as Use Cases, I* and KAOS goal models, Goal/Strategy maps, Entity-Relationship diagrams, and Workflow models.

1 Introduction

In nowadays business competitive world, organizations have recognized the need for more agility in the development of their Information Systems (IS). Indeed, it is not anymore enough to have a system that fulfils the needs of a business. Now, it is necessary that systems evolution matches the evolution of businesses. [Salinesi03a] [Salinesi04].

According to [Jarke94], a system evolution can be designed as the movement from a situation to a new one. Traditionally, these situations are (as shown in Fig. 1) respectively captured in *As-Is* models and *To-Be* models. In many academic approaches, the evolution requirements are expressed by only specifying the To-Be. Our experience in several industrial projects showed us that, on the contrary, evolution requirements were initially captured relatively to the As-Is (even when this one is implicit and not specified), then the To-Be models are specified (if necessary from scratch). In this approach, the change process is an As-Is to To-Be movement for which requirements can be expressed as gaps and similarities relative to As-Is models. In [Salinesi03a], we demonstrated that this framework can be adapted to four different classes of IS evolution projects, namely: direct change propagation, customisation from a product family, adaptation of a baseline product and component assembly.

A. Persson and J. Stirna (Eds.): CAiSE 2004, LNCS 3084, pp. 338–352, 2004.

- In *direct change propagation*, the issue is to propagate the change requirements from the business level to the system functionality level. Change requirements are expressed as gaps with the As-Is [Salinesi03a].
- In *customisation from a product family*, the issue is to match the initial vision of the business (defined in As-Wished models) with models of the functionality capability of the product family (defined in Might-Be models) to specify the To-Be on the business and on the system functionality levels. The requirements for these To-Be models are expressed as similarities with the As-Wished and with the Might-Be [Zoukar04a] [Zoukar04b].
- In the case of *adaptation of a baseline product*, the issue is to find how the To-Be should differ from the As-Wished (on the business level) and the Is-Baseline (on the system functionality level) to obtain a correct adaptation of the baseline product. The required differences are specified under the form of gaps [Rolland04].

In *component assembly*, the change process consists in retrieving from a collection of COTS those that match the organization needs (defined in As-Wished business models (BMs)), and assembling them to obtain the To-Be situation. In this complex process, the matching requirements are expressed as similarities between the As-Wished BMs and the Might-Be models of the system functionality (SFMs). The requirements for component adaptation and assembly are expressed as gaps with the initial Might-Be [Rolland01b].

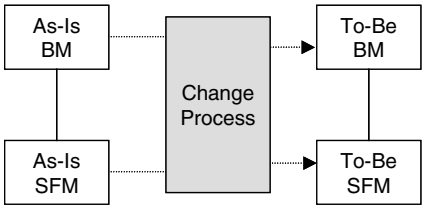

Fig. 1. Methodological framework for IS evolution

Despite the diversity of engineering processes dealing with these four classes of IS evolution projects, our experiences led us to identify two common underlying strategies. One is based on gaps, and the other is based on similarities. Intuitively, our proposal is to express IS evolution requirements with:

– gaps as operators that express transformations of As-Is models into the To-Be models, and
– similarities specify through predicates what the As-Is (or Might-Be) and To-Be (or As-Wished) should have in common.

Our languages of gaps and similarities are defined as two generic typologies of gap operators and similarity predicates. A number of gaps operators and similarity predicates were discovered within industrial projects. To achieve genericity and completeness, the typologies were specified so that the gap operators and similarity predicate would apply on the elements and structures of a generic meta model. This generic meta-model can be instantiated by any specific meta-model such as Use Case, Entity Relationship, etc.. As gap operators and similarity predicates relate to the generic meta-model, the specific elements and structures identified for a specific meta-model can be easily transposed onto specific gap operators and specific similarity predicates.

This allows us to express gaps and similarities in a specific way whichever language are required in the IS evolution project to specify the As-Is, As-Wished, Is-Baseline, Might-Be or To-Be models. The language formed by our two typologies can then be used to express evolution requirements; there is no assumption whether or not expressing these requirements necessitates the existence of the As-Is, As-Wished or Might-Be models. Our experience showed that this language is richer and more allows more precise specification of evolution requirements than the languages that are intuitively used in practice or developed in academy without a reference meta-model.

The rest of the paper is structured as follows: section 2 details the approach adopted to develop our generic typologies of similarity predicates and gap operators. The two resulting languages are respectively described in section 3 and 4. An example of application with goal/strategy maps, E/R diagrams and workflow models is presented in Section 5. Sections 6 and 7 discuss respectively related works and the future works in our research agenda.

2 Approach Taken to Develop the Typologies of Gap and Similarity

Gaps and similarities are the two central concepts needed to express requirements in an IS evolution project. There are different kinds of gaps, different kinds of similarities, and those can be defined to express requirements related to different kinds of models. Therefore, we adopted a systematic approach aiming to (i) identify a list of gap operators and similarity predicates that would be as complete as possible, and (ii) provide the means to adapt the identified gap operators and similarity predicates to the project situation.

2.1 General Overview

The general overview of our approach is presented in Fig. 2. As the figure shows, gaps (represented by the symbol Δ) and similarities (represented by the symbol $\equiv$) are specified at the modelling level. Gaps and similarities are relative; hence they can relate to various models (As-Is BM, As-Is SFM, As-Wished BM, Might-Be SFM, etc). These models can be specified using different meta models such as Use Case, E/R, Workflow, Business Process, Goal hierarchy, etc. In a concrete project, it is for instance possible to express a number of change requirements using gaps predicates, then build the To-Be models, then forecast the value/cost ratio of the change requirements in reference to the future business and system.

The link between models and meta-models is an instantiation link. This means that any element in a model instantiates an element in a meta-model. Similarly, we believe that any requirement expressed as a gap or as a similarity on the modelling level should be an instance of some concept formalised on the meta-model level. We call the different kinds of meta-models "specific meta models" (as they all have their specificities), and the typologies of gap operators and similarity predicates that correspond to them "specific typologies". The link between the specific typologies and the specific meta-models shows that any specific typology of gap operators or specific typology of similarity predicates applies on a specific meta-model.

Rather than defining as many typologies of gap operators and similarity predicates as there are of specific meta-models, our approach proposes to take a larger generic view. A generic meta-model level is thus used on top of the specific meta-model level. This generic meta-model level contains a unique generic meta-model on top of the specific meta-models, and a generic typology of gap operators and of similarity predicates on top of the specific typologies.

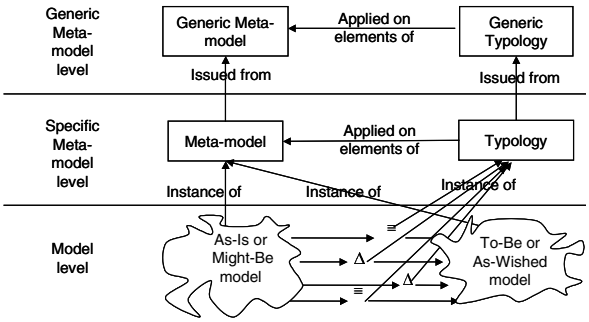

Fig. 2. Overview of the approach for defining the typologies

Let us take the example of a hotel room booking system specified with a Use Case Model. Similarities could be used to express which Use Case, and which part of the Use Cases have a similar equivalent in the Use Case Models defined for a number of software packages available for the hotel business. These similarities instantiate the specific typology of similarity predicates developed for Use Cases. This typology contains predicates such as "Two actors have the same name" or "the attributes of a Use Case include those of another Use Case", etc. These predicates shall be used to express requirements such as: (i) the actors in a Use Case have the same name as those identified by the legacy system, and (ii) a component of the software package can be selected if its attribute values are included in the attributes values defined for one of the Use Cases that define the legacy system.

Similarly, the specific typology of gap operators contains the operator "Add Use Case", "Change Origin of Use Case-Actor Association", or "Merge Actors". This allow to express requirements such as: (1) add a "cancel booking after booking date" Use Case in the Use Case Model of a booking system, or (2) merge the "salesman" and "receptionist" actors into a unique "front-desk" actor to simplify the organisation of sales in the hotel.

As these examples show it, specifying gaps and similarities to express requirements is not difficult once the specific typologies of gap operators and similarity predicates are known. However, defining these typologies from the generic typologies requires knowing how the specific meta-model at hand specialises the generic meta-model.

2.2 Generic Meta-model

The generic meta-model is not a 'universal' meta-model that would aim at specifying any concept in any method. On the contrary, its purpose is only to make explicit the

main elements and structures of method parts that can be specified in a product meta-model [Rolland02]. We developed this meta-model based on a long experience of meta-modelling and meta-meta-modelling [Grosz97] [Plihon96] [Si-Said99] [Ralité01]. The generic meta-model shown in Fig. 3 with the UML notation stipulates that any that can be represented graphically model is composed of a collection of elements with properties.

Elements have a name and have properties. For example, a Use Case Model, a Use Case, an Actor, or a Scenario are different elements of the Use Case meta-model. The various attributes of Use Cases are as many properties that directly relate to the Use Case element. Elements are also classified into two pairs of sub-groups. First, a distinction is made between simple elements and compound elements. Second, elements are classified into link and not link.

Compound elements are composed of elements. Those can at their turn be simple or compound, and thus several levels of composition can be defined. For example, a Use Case Model is composed of Use Cases, which are at their turn composed of scenario descriptions, etc. Let us notice that any model or diagram is composed of elements. There are models in which one element always appears. This is for example the case of the system boundary in a Use Case model, or for the Object class in a class inheritance diagram. These elements are classified as *Root*.

Link Elements are connectors between pairs of elements. Every link can be oriented. Therefore, one of the linked elements plays the role of *Source* and the other of *Target*. For example, the "extends" relationship and the uses relationships are link elements of the Use Case meta-model.

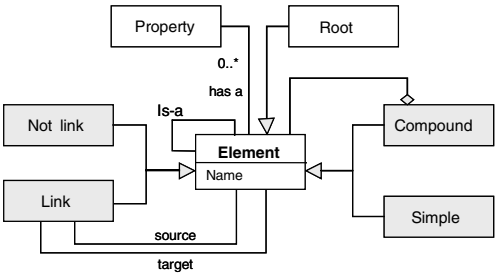

Fig. 3. Generic meta-model for defining the gap and the similarity typologies

The systematic definition of generic gap operators and generic similarity predicates directly results of the structure of the generic meta-model. For example, (i) adding or removing elements in the composition of a compound element are gaps, and (ii) having the same collection of components is a similarity that typically relate to compound elements. The two typologies were therefore developed by: first, looking for gap operators and similarity predicates in the literature, then, by systematically generalising them by applying them on all parts of the generic meta-model.

3 Generic Typology of Similarity Predicates

The generic meta-model indicates that any meta-model is composed of elements with properties. Besides, the structure of a meta-model is shown through element composi-

tion and through links between elements. Based on this, the generic typology of similarity predicates emphasises that given a pair of elements, (i) their properties can be similar, and (ii) their structure can be similar. As Fig. 4 shows, there are thus two classes of similarities, intrinsic similarities and structural similarities.

A pair of elements has an *intrinsic* similarity if they have similar properties. Element properties can be considered similar if they have a close semantics. In the first place, intrinsic similarity relates to synonymy. However, hyponymy (or the other way round hyperonymy) are also semantic relationships that can be used to define intrinsic similarities.

Structural similarity deals with (i) the composition of elements, and (ii) their organisation within models. There are thus two classes of structural similarity: compositional similarity, and relational similarity. Contrary to intrinsic similarity that only involves the two compared elements, structural similarities also imply comparisons between other elements that are related to the two compared ones. As shown in Fig. 4 by the aggregation link from structural similarity class to the similarity class, a structural similarity is a complex one and involves other similarities. For example, two elements have the "same components in a composition" if each component in one element has a semantically "same" counterpart in the composition of the other element.

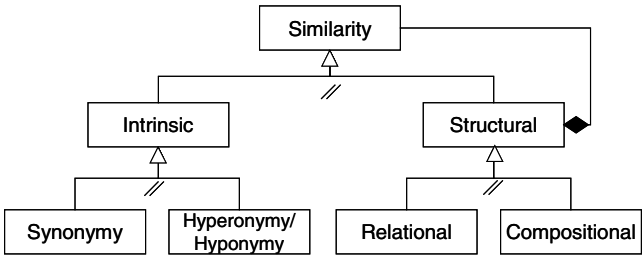

Fig. 4. Generic typology of similarity predicates (main categories)

Thirty-three similarity predicates composing the generic typology have been identified so far and are listed in table 1. These were classified in the four aforementioned classes and are defined as follows:

(i) *Synonymy*:

- Two elements have a synonym type if their types are equal or have a common super-type (they are then cousins). This is, for example, respectively the case of two goals in map models, or two actors in Use Case models.
- There are different degrees of possible resemblances between the properties of a pair of elements: two elements have the *same property* when their properties have exactly the same name and the same meaning (for example two extension conditions in two different Use Case models); they have *alike properties* when their properties are identified with different words but have the same meaning (for example two classes that specify the same business object in two different ERP modules); or they have a *resembling property* when the properties have different names and values, but they still have a close meaning (like for example a standard business object in two different ERPs).

(ii) *Hyponymy/Hyperonymy* relates two elements when the meaning of the one subsumes/is subsumed by the meaning of the other. As with synonymy, hyponymy/hyperonymy similarity can be defined on the type and on the properties of elements:

- with respect to type, hyponymy/hyperonymy relates to a *father/son* relationship between the types of the involved elements.
- With respect to properties, two elements are in a hyponymy/hyperonymy relationship if the properties of the ones *includes/extends* the properties of the other. This is for instance the case when the attributes of one class are included (or have a similar counterpart) in the collection of attributes of another class.

(iii) *Relational* similarities are defined between link elements that are connected to similar source/targets, or between elements that are related to the rest of their models through similar links. As table 1 shows, there are different kinds of relational structure similarity predicates. These include (without being restricted to): *same number of links* (when two elements are source/target of the same number of links)*, same number of links entering in a node* (when two elements are source of the same number of links)*, same number of links outgoing from a node* (idem, the other way round)*, same/alike/resembling source, target, or source and target* (when two links have similar extremities)*, same depth* (same max distance between nodes and leaves of the trees they belong to) or *same height* (same max distance between nodes and the root of the trees they belong to).

(iv) *Compositional* similarities deal with compound elements that are similar in their composition, and with elements that belong to similar compositions. Table 1 quotes a number of compositional structure similarity predicates: *same cardinality of a component* (when two compound elements have the same number of components), *same / alike / resembling components in a composition* (when the compositions of two compound elements are comparable), *same/alike/resembling common component in a composition* (when part of the compositions are comparable), *etc*.

Table 1. Generic typology of similarity predicates (details)

Synonymy	Hyperonymy Hyponymy	Relational	Compositional
Type	Type	Same number of links	Same cardinality of a component
Equal type	Father type	Same links number entering in a node	Same/Alike/Resembling components in a composition
Cousin type	Son type	Same links number outgoing from a node	Same/Alike/Resembling common component in a composition
		Same/Alike/Resembling source	Part of Same/Alike/Resembling compound
Property	Property	Same/Alike/Resembling target	
Same property	Includes property	Same/Alike/Resembling source & target	
Alike property	Extends property	Same depth	
Resembling property		Same height	

4 Generic Typology of Gap Operators

We propose to define gaps operators under the form of change operations made on models. There are different kinds of such operations: adding elements, removing them, changing their definition, replacing them by others, etc. Fourteen operators have been identified and defined on the generic level, i.e. to apply on the generic

Table 2. Meta-model elements and related operators

Element	Link	Compound	Property	Root
Rename	ChangeOrigin	AddComponent	Give	Add
Merge		RemoveComponent	Withdraw	Remove
Split		MoveComponent	Modify	
Replace				
Retype				

meta-model. Each operator identifies a type of change that can be performed on an element or a property of the As-Is model. Table 2 sums up the *generic gap typology* composed of 14 operators that we identified on *Element* or *Property*.

The operators have been classified according to the part of the generic meta-model on which they apply. The five operators that can be applied on an element can also be applied on any of the element specification (i.e. Link Element, Not Link Element, Compound Element and Simple Element). The *Rename* operator changes the name of the element in the To-Be model. Two separate As-Is elements become one in the To-Be model when the *Merge* is applied on them. For example, two Use Cases can be merged into one to indicate that from now on, the corresponding services shall be provided by the system within a single transaction. In the opposite, the *Split* operator decomposes one As-Is element into two To-Be elements. For example, a Use Case UC1 can be split into UC2 and UC3. This can occur when the user requires to be able to use independently the service defined in UC2 and UC3, and initially defined as part of UC1. It may be necessary to replace an As-Is element by a different To-Be one with the *Replace* operator. The *Retype* operator allows to change in the To-Be model the type of an As-Is element.

All the other operators can only be applied on one type of element: the *ChangeOrigin* operator only applies on Link elements in order to change the sources or targets of links. The changeOrigin operator can for instance be used to specify that the initiator actor of a Use Case has changed. The *AddComponent* operator is used when a component is added in the To-Be. In the opposite, a component can be removed with the *RemoveComponent* operator.

Three operators were also defined to specify when it is the properties of elements that change: the *Give* operator allows to add a property to the To-Be element. This operator is for example used when a new invariant predicate is attached to a Use Case. In the opposite the *Withdraw* operator removes an As-Is property in the To-Be element. Finally, the *Modify* operator changes the property of the To-Be element.

Finally, two operators can be applied on Root elements: Add, that allows to create a model by introducing the Root, and the other way round, the Remove operator when the model is destroyed. Typically, the system boundary is the first element to be added; and the last element that should be removed when a Use Case model is created or destroyed. Each gap operator at the specific meta-model level is defined with parameters to specify on which element it is applied.

5 Example of Application

This section illustrates the ability of our approach to adapt to different contexts. The example taken is that of an IS evolution project in which goal/strategy maps [Rol-

land99] [Rolland01a], E/R diagrams [Chen76], and workflow diagrams [Casati96] are used. The application domain is the one of hotel room booking [Salinesi03b]. In this example, a hotel chain initially uses a system to handle room booking in a centralised way. A project is undertaken to change the hotel booking business process in order to improve competitiveness.

In the current situation, the products offered by the hotel chain are independently designed in the system in a flat list which is augmented each time a new product is designed. Once products are in the list, they are used to create booking contracts. Any product of this list can be removed when it terminates its lifecycle. As shows the goal/strategy map extract on top of Fig. 5, there are two strategies available to achieve booking contracts management goal in the current situation: on the spot (i.e. at the hotel), and with a third party (e.g. at the city's tourist office or in an agency). The contract management process ends up either by cancellation of the contract, or by consumption of the associated product by the consumer.

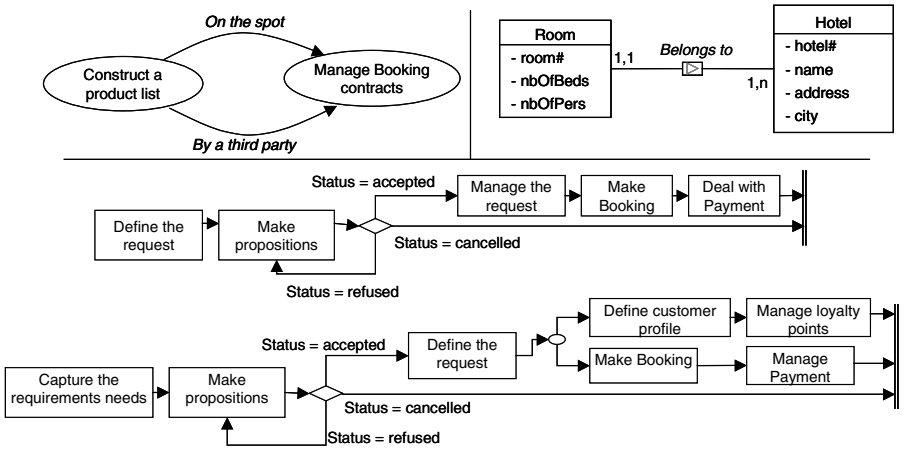

Fig. 5. Extracts of the three As-Is models; goal/strategy map (top left); E/R (top right); workflow (bottom)

A number of evolutions were required. Three major evolutions can be highlighted: (1) From now on, the system should be customer-centric; (2) It should be possible to propose complex products (such as packages including tourist activities) to customers; (3) The sales channels have to be diversified.

Each of the three following sub-sections shows how specific typologies of gap operators and similarity predicates are defined then used to express evolution requirements with each of the three modelling techniques used in the project.

Two different ways have been chosen to manage these evolutions: (i) by modification of the legacy to create an 'in-house' To-Be (ii) by introducing COTS. These two approaches are simultaneous described in following sub-sections; the first one is based on gaps whereas the second one uses similarities.

5.1 Expressing Goal/Strategy Maps Evolution Requirements

A goal/strategy *map* is an oriented graph which nodes are *goals* and edges s*trategies*, i.e. ways to achieve a goal. Instantiating the generic meta-model shows that goal/strategy maps are compound elements that contain "sections". Every section in a map is itself a triplet composed of two goals and a strategy. One goal plays the role of source and should be achieved for the section to be undertaken. The other goal is the target i.e. the section aims at achieving. Strategy is a link between goals that defines way to reach the target goal from the source goal. Goals are simple/not link elements which main property is the goal statement structure [Ralyté01]. As shown in [Rolland04], this allows to define specific operators for each kind of elements in goal/strategy maps.

For example, the diversification of sales channels calls for a change on the As-Is goal/strategy map (Fig.5) in which the As-Is system only proposes to achieve the "Manage Booking contracts" goal with two strategies: *on the spot* and *by a third party*. adding a third strategy. The *AddStrategy(with web site, Attract People, Manage Customer relationship)* gap operator can for example be used to express this requirement. It is a specialisation of the *AddComponent* gap operator.

Another decision could be to use a hotel management software package (e.g. such as Orchestra, WebHotel or Betisoft). Fig. 6 shows the intended business goals and strategies and the facilities provided by one of these COTS. The COTS models have a number of structural and intrinsic similarities with the As-Wished models, namely: (i) the two goals "Attract people" and "Attract potential clients" are synonymous, and have alike properties, and (ii) the COTS strategy of "Promotion" is labelled as a hyperonym of the strategy "By keeping customer's loyalty" that was initially wished. It is therefore decided to acquire the COTS and implement it in the new system.

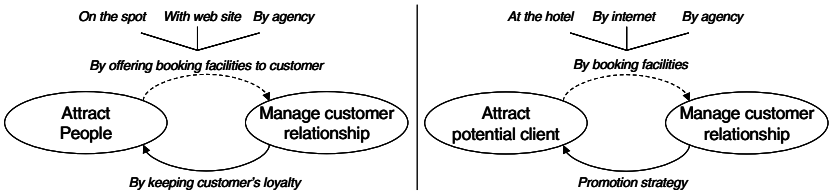

Fig. 6. Extracts of As-Wished (left hand) and COTS (right hand) goal/strategy maps

5.2 Required Evolutions with Respect to the WIDE Workflow Model

The contracting process is currently achieved as described in the WIDE workflow model in Fig. 5. Fig. 7 shows two other workflow models, one which is the wished target defined by the stakeholders, and the other which is the one supported by the selected COTS.

In the WIDE meta model [Casati96], a *Workflow Schema* is a graph which nodes are *Tasks* and edges *Connectors*. Connectors are links between tasks that define the order in which they must be executed. Besides, a set of *Variables* with values (that can be a default value) is associated with any Workflow schema. A *Task* is a unit of work. Every Task has a *ConditionFonction* specifying the conditions that need to be

satisfied before the task can be executed. A *Null Task* is a task that immediately finishes after it starts (no work is done); it is introduced only as a conceptual device to define the semantics of a workflow schema. For example, an empty schema is defined as containing a null task. A *Connector* defines a link of precedence / succession between tasks. There are different kinds of connectors: fork, and join.

The requirements for the new organisation have been initially defined as follows: the system should be customer centric. It is thus decided to *rename* the first task "Define the request" into "Capture customer needs". Stakeholders also decided to enforce customer loyalty. Two tasks should be *added* for this purpose: (i) "Define Customer profile" that will allow the hotel consortium to make personalised offers to clients; and (ii) "Manage loyalty points" that specifies that each time the client buy a product, it receives loyalty points. Finally, the task "Deal with payment" is *replaced* by "Manage payment". Indeed, the payment can not only be made with credit card, cash, personal cheques etc. as before, but henceforth also with loyalty points.

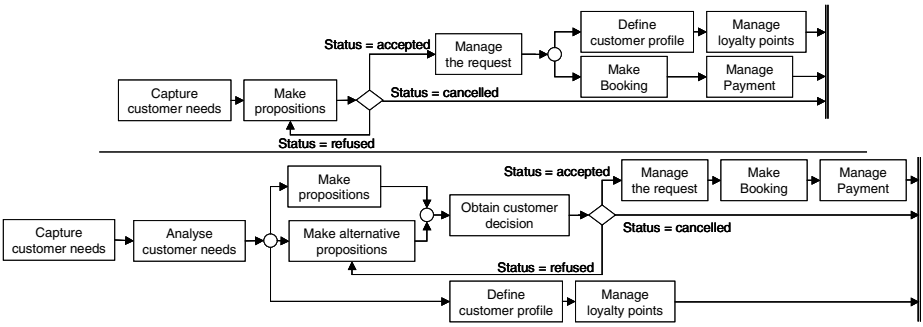

Fig. 7. Parts of the As-Wished model (top) and the Might-Be model (bottom) concerning the products in the catalogue

A number of the facilities can also be implemented by adapting the COTS that was considered in the previous section. Indeed, the COTS Might-Be model has the *same* decision function as the As-Wished discussed above. Besides, a number of the tasks it supports have the *same properties as* the ones that were initially wished, e.g. "manage the request" and "manage loyalty points". Compositional and relational similarities can be easily detected too. All the facilities offered by the COTS and specified in the Might-Be are adopted as fulfilling the requirements that were initially wished. Therefore, the decision that is made is to keep all these facilities. This requirement for the To-Be is therefore specified under the form of similarities with the COTS-supported workflow.

5.3 Evolution Requirements with Respect to the E/R Models

One of the important required evolutions was to replace the flat product list with a collection of complex product definitions. It is decided that contract should now include all the hotel facilities such as for instance tennis, swimming pool, amphitheatre and meeting rooms, Internet connections. In terms of gaps with the As-Is E/R model,

the evolution requirements are thus to: *AddEntity(Activity), AddEntity(Option), AddRelationship(Proposes, Hotel, Activity), AddRole(<Proposes, Hotel, Activity>), AddRelationship(Offers, Hotel, Option), AddRole(<Offers, Hotel, Option>), AddAttribute(Activity, activity#), AddAttribute(Activity, activityName)*. These gaps directly instantiate the specific typology of gap operators developed for the E/R meta-model. The E/R model resulting from these evolution requirements can be designed as shown in Fig. 8.

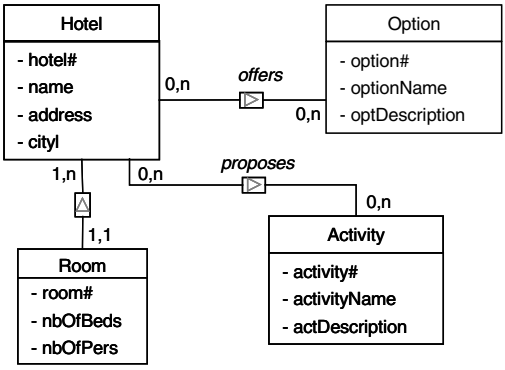

Fig. 8. Extract of the To-Be model with the products catalogue

Fig. 9 shows an E/R model of the COTS. A number of structural similarities are necessary to confirm that the COTS matches the wished requirements. For example we can notice that there are type and property synonymies concerning each entity of the To-Be model with their counterpart in the Might-Be model. The "same common component in a composition" structural compositional similarity allows to show that the part of the To-Be is included in the Might-Be model.

These structural and intrinsic similarities between the two models help the requirement engineer to master the matching process in order to establish the COTS customisation tables. In addition, we can notice that the E/R model of the COTS is richer than the To-Be.

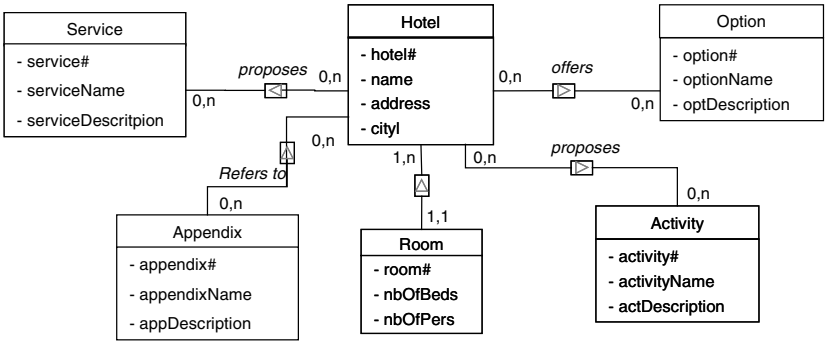

Fig. 9. Parts of the Might-Be model concerning the products in the catalogue

6 Related Works

Handling IS evolution is an important issue in both the academic world and in industry; as show for example the IWPSE series of workshop [IWPSE]. The literature proposes different approaches to manage IS evolution. Some approaches deal with the propagation of change on the system implementation using a maintenance or a correction point of view. For example, [Breche96] defines a typology of operators to make the class instances migrate from the old system to the new one. [Sadiq00] and [Bandinelli93] propose similar approaches, respectively with a workflow meta-model and a software process meta-model. Our approach differs from those in that we adopt a requirement-driven point of view [Rolland04], whereas the aforementioned approaches rather focus on technical aspects such as system implementation or instance.

Several typologies of gap operators or similarity predicates were already proposed to maintain the consistency and the validity of models [Breche96], [Bandinelli93] [Deruelle99], [Ralyté01]. However, each of these typologies is only defined for one specific meta-model. In our approach, we propose a generic typology of gap operators and a generic typology of similarity predicates that can be specialised for any meta-model, as we showed with 3 different examples of application and in [Etien03].

Similarity measurement is also a topic of interest in different areas of IS engineering and Requirements Engineering. For example, [Castano92] proposes to evaluate components reusability through conceptual schema. [Jilani97] used similarity measures to select best-fit components. Similarity metrics for heterogeneous database schema analysis were introduced by [Bianco99]. Our similarity approach is inspired by [Castano92] and [Bianco99]. It could be compared to that of [Ralyté01], except that we are not defining similarities between meta-models but between models, and except the fact that the purpose is not just to find which element looks like another, but also to specify evolution requirements according to which there should be similarities between a future situation and an old one. Similarity measurement can also be automated (e.g. see [NattOchDag01]). Such techniques could be used to guide the matching between COTS and As-Wished models, but manual work is still needed to transform the matching results into proper evolution requirements.

7 Conclusions

The example of the hotel room booking system shows how to use the gap modelling and similarity modelling to express requirements in a context of IS evolution. Applying this approach on three different meta-models does not demonstrate that the approach is generic. However, combined with the fact that we already used this approach in the context of several different industrial projects ([Zoukar04a], [Rolland04], [Salinesi02a], [Salinesi02b]) we believe it is sufficient to show that this approach is indeed *usable* in different methodological contexts, and *scalable* to real-world projects. Our approach to deal with scalability is to abstract As-Is, As-Wished, Might-Be and To-Be using goal models, then drive the analysis in a top-down way. As shown in the aforementioned experience reports, this helps to undertake the analysis in a synthetic way, prune uninteresting parts of the business and of the system functionalities, then concentrate on those parts of the business that are the most likely

to change or with respect to which stability is crucial. Our approach is however not applicable in any IS evolution project. For instance it shouldn't be used in project in which the foreseen change has a revolutionary impact on the IS and on its business environment (i.e. there should be only a limited amount of evolutions).

Further evaluations of our approach are however needed to substantiate our claim. Besides to being generic, we expect that the evolution requirements language that is constituted by our typology of gap operators and by our typology of similarity predicates has also a number of other qualities such as completeness, exhaustiveness, minimality, concision, and coherence. We have already empirically evaluated the gap typology according these criteria [Etien03]. However, we believe further experiments are needed, e.g. to evaluate the efficiency of expressing evolution requirements using our approach, and to compare it with other approaches in real project situations.

Another important issue is the one of guiding the elicitation of evolution requirements and checking their correctness. We are currently developing three process models. One is to elicit compliance requirements ensuring an adequate transition to the new system when business evolution requirements have already been specified [Salinesi03b]. The second one is being developed for an ERP project at the French national railway company. It aims at guiding the elicitation of ERP requirements [Zoukar04a] [Zoukar04b] so as to ensure maximum fitness of the ERP implementation with the new organisation of the business that the ERP project makes itself evolve. The process model was developed in a project with a French company of the automotive industry and guides adaptation of a baseline product [Rolland04]. We would like to develop in the near future a process model for the fourth kind of project that our methodological framework lead us to identify (namely component assembly), and to look for an integrated multi-way-of-working process model [Plihon96] [Grosz97] [Ralyté01] that could be adapted to any project situation. We believe that one of the salient characteristic of these process models might be that evolution requirements are not independent from each other. Clusters of change requirements and inter-requirements dependency links are concept that we are considering in our current research project to complete our approach.

References

[Bandinelli93] Bandinelli, S., et al. *Software Process Model Evolution in the SPADE Environment*. IEEE Transactions on Software Engineering, 19(12) pp.1128-1144, (1993).

[Bianco99] Bianco G. *A Markov Random Field Approach for Querying and Reconciling Heterogeneous Databases*. Proc. DEXA'99, Pisa, Italy, September 1999.

[Breche96] Breche P. *Advances Primitives for Changing Schemas of Object Databases*, Proc. CAiSE'96, Springer Verlag (pub), Heraklion, Greece, 1996.

[Casati96] Casati, F., Ceri, S., Pernici, B., Pozzi, G. *Workflow Evolution*. Proc. Of 15th Int. Conf. On Conceptual Modeling (ER'96), Cottbus, Germany, pp. 438-455 (1996)

[Castano92] Castano S. De Antonellis V. Zonta B. *Classifying and Reusing Conceptual Components*. Procs. of ER'92, pp. 121-138, Karlsruhe, 1992.

[Chen76] P. Chen. *The Entity-Relation Model - Towards Unified View of Data*. ACM Transactions on Database System, 1(1):9-36, March 1976.

[Deruelle99] Deruelle, L., et al. *Local and Federated Database Schemas Evolution An Impact Propagation Model*. Proc. DEXA'99, pages 902-911, Italy, September 1999.

[Etien03] Etien, A., Salinesi, C. *Towards a Systematic Definition of Requirements for Software Evolution: A Case-study Driven Investigation*. Proc of EMMSAD'03, Austria, 2003.
[Grosz97] G. Grosz, et al. *Modelling and Engineering the Requirements Engineering Process: An Overview of the NATURE Approach*. Requirements Engineering Journal, (2), 1997.
[IWPSE] IWPSE International Workshop on the Principles of Software Evolution
[Jarke94] Jarke, M., Pohl, K. *Requirements Engineering in 2001: Managing a Changing Reality*. IEEE Software Engineering Journal, pp. 257-266. November 1994.
[Jilani97] Jilani L.L. Mili R. Mili A.. A*pproximate Component Retrieval: An Academic Exercise or a Practical Concern?*. Procs. (WISR8), Columbus, Ohio, March 1997.
[Plihon96] Plihon V. *Un environnement pour l'ingénierie des méthodes*. PhD Thesis, University of Paris1 Panthéon-Sorbonne, 1996
[Ralyté01] Ralyté J. *Ingénierie des méthodes à base de composants*, PhD Thesis, University of Paris1 Panthéon-Sorbonne, January 2001.
[NattOchDag01] Natt och Dag J. *Evaluating Automated Support for Requirements Similarity Analysis in Market-driven Development*. Procs. REFSQ'01, Switzerland, 2001.
[Rolland99] Rolland C., Prakash N., Benjamen A. *A Multi-Model View of process Modelling*, Requirements Engineering Journal, Vol 4, pp 169-187, 1999.
[Rolland01a] Rolland C., Prakash N. *Matching ERP System Functionality to Customer Requirements,* Proceedings of RE'01, Canada, pp. 66-75, 2001.
[Rolland01b] Rolland C. *Requirements Engineering for COTS based Systems*. XXVII Latin American Conference on Informatics (CLEI'2001), Merida, Venezuela. September, 2001.
[Rolland02] Rolland C. *A Comprehensive view of Method Engineering* Invited talk, PromoteIT2002, Knowledge Foundation Symposium, Skovde, Sweden, April 2002.
[Rolland04] Rolland, C., Salinesi, C., Etien, A. *Eliciting Gaps in Requirements Change*. Requirement Engineering Journal Vol. 9, pp1-15, 2004
[Sadiq00] Sadiq, S. Handling *Dynamic Schema Change in Process Models*. Australian Database Conference, Canberra, Australia, 2000.
[Salinesi02a] Salinesi, C., et al. *A Method to Analyse Changes in the Realisation of Business Intentions and Strategies for Information System Adaptation.* Proc. EDOC'02, Sept. 2002.
[Salinesi02b] Salinesi, C., Wäyrynen J.: A Methodological Framework for Understanding IS Adaptation through Enterprise Change. Proceedings of OOIS'02, France, September 2002
[Salinesi03a] Salinesi, C., Rolland, C.: F*itting Business Models to Systems Functionality Exploring the Fitness Relationship*. Proceedings of CAiSE'03, Velden, Austria, 2003.
[Salinesi03b] C. Salinesi, A. Etien, *Compliance Gaps: a Requirements Elicitation Approach in the Context of System Evolution*, Proceedings of the OOIS'03, Switzerland, Sept. 2003.
[Salinesi04] Salinesi C. et al *Goal / Strategy Maps - Methods, Techniques and Tools to Specify Requirements in Different Evolutionary Contexts*. Proc. INCOSE'04, France, June 2004.
[Si-Said99] Si Said S. *Proposition pour la modélisation et le guidage des processus d'analyse des systèmes d'information*. University of Paris1 Panthéon-Sorbonne, February 1999.
[Zoukar04a] Zoukar I., Salinesi C. *Engineering the Fitness Relationship between an ERP and the Supply Chain Process at SNCF*. Proc. IRMA'04, USA, May 2004.
[Zoukar04b] Zoukar I., Salinesi C. *Matching ERP Functionalities with the Logistic Requirements of French Railway*. Proc. ICEIS'04, Portugal, April 2004.