ReeF: Defining a Customizable Reengineering Framework

Gemma Grau and Xavier Franch

Universitat Politècnica de Catalunya (UPC)
c/ Jordi Girona 1-3, Barcelona E-08034, Spain
{ggrau, franch}@lsi.upc.edu

Abstract. During their life span, organizations must adapt continuously to an always evolving context and so have to do their Information Systems and the processes around them. The scope of these changes ranges from small-scale maintenance modifications or the redefinition of some business processes to the full deployment of a new system. In all cases, the resulting Information System will seldom be built from the scratch; as even when deploying it for the first time, we may consider that it starts from the description of the current human processes. For that reason, we may consider Information System development and its evolution as a reengineering process. In this paper, we present a framework that defines the generic activity of reengineering using Method Engineering techniques. The framework is built upon existing reengineering methods from different disciplines and provides six generic phases that can be instantiated with the purpose of defining new reengineering methods.

Keywords: Method Engineering, Reengineering Framework, Business Process Reengineering, Software Process Reengineering, *i** Modelling and Analysis.

1 Introduction

Information Systems (IS) are in continuous change for various reasons. Changes that affect the system over time include requirements, technology and business processes [30]. All these changes are diverse in nature and may require different treatments according to their impact over the IS. On the one hand, the current software may have to be rebuilt, in order to create a product with added functionality, better performance and reliability, and improved maintainability [26]. On the other hand, if the changes on the business are too profound, a new IS may have to be deployed by adapting an already existing legacy system or by building a new one. Therefore, in all these situations, there are processes, artefacts and knowledge that can be taken as a starting point.

According to [31], traditional reengineering activities include: identifying, delineating, and modelling the existent process; analysing it for deficiencies; proposing new solutions; and implementing the new design in terms of new technical systems and new organizational structures. It is possible to observe that most of the methods proposed for the specification, development or acquisition of IS already support some of these activities. For instance, some IS methods explicitly mention the term reengineering in their proposal, as in [1], [2], [3], [4], [20], [21], [26], [30], [32].

On the other hand, some other methods not defined in the reengineering context, tackle with some of those activities, among them we mention [7], [8], [10], [11], [16], [17], [19], [24], [28]. Therefore, we may consider that changes on ISs are all part of a reengineering activity, which supports our claim of IS development being treatable similarly to IS reengineering.

An observation that can be made is that each above-mentioned reengineering-related approach focuses on a particular discipline: business processes [2], [20]; software architectures [3], [26]; or software platforms [4], [32]. Despite of this diversity, there are a lot of similarities when the methods are deeply analysed. Actually, some of the differences lie more in the detail (e.g., using this or that technique) than in the rationale or the rough reengineering process. However, in some proposals, some of the reengineering activities and artefacts are not mentioned and the lack of a generic framework makes difficult to apply them through a complete reengineering process.

In order to address this problem, we propose ReeF, a customizable **Ree**ngineering **F**ramework which is based on the principles of Method Engineering [5], [23], [25], [28] with the aim of assisting on the construction of new processes based on the existing ones. ReeF has been built in two steps: first, abstracting the phases and method artefacts from the existing method PR*i*M, a Process Reengineering *i** Method [13], by using the Approach for Method Reengineering [28]; and second, generalizing the obtained phases and method artefacts by analysing other existing reengineering techniques from different domains [1], [2], [3], [4], [7], [20], [21], [26], [30], [32]. Once obtained and validated, we show an example of framework customisation by defining SAR*i*M, a method for software architectures reengineering based on *i**.

The benefits resulting from this process are twofold. In the one hand, the definition of ReeF may help to understand, reconcile, and analyse existing reengineering methods, and also to formulate new specific ones. With this aim, ReeF clearly establishes the reengineering phases and the method artefacts involved in each phase (techniques needed, modelling languages used, tool support provided, and roles). On the other hand, the abstraction and generalization mechanisms used for abstracting and generalizing ReeF from other methods (such as PR*i*M), may be applied to generate other customizable frameworks based on a different development point of view (as we have done with process reengineering).

The rest of the paper is organized as follows. In section 2 we outline the research method followed to define the framework. The PR*i*M method, upon which ReeF is based, is presented in section 3. The proposed framework is detailed in section 4 and customized in section 5 for obtaining SAR*i*M. Finally section 6 presents the conclusions and future work.

2 Research Method

The main purpose of this research is to define a generic framework in which existing reengineering techniques can be reconciled, adapted and analysed. As a result, new reengineering methods in different disciplines and domains can be created by derivation and combination of reusable fragments. As a result, this work is related with Method Engineering, which is the discipline that constructs new methods from parts of existing methods [5]. There are several proposals that address Method Engineering [5], [23], [25], [28], among which we remark:

- The OPEN Process Framework (OPF) [25] is a generic framework that provides a repository with a wide range of Method Components, which are different parts of existing methods described at different levels of detail that can be used for defining other methods in different domains. A Method Component can be specialized into Endeavour, Language, Producer, Stage, Work Product or Work Unit, which, in turn, can all be specialized forming a complete hierarchy of elements. The OPF repository of Method Components is very complete, thus enabling the selection of those components more suitable for the specific purposes of the method.
- The Approach for Method Reengineering [28] proposes a bottom-up process for transforming already existing methods into several pieces of *method chunks* which are stored in a *method base*. From the stored *method chunks*, assembly-based construction of methods is done by applying the following three steps [23]: method requirements specification, *method chunks* selection and *method chunks* assembly.

We have considered using the OPF approach for generating ReeF; more precisely we have studied the customizations for a Business Reengineering Project and for a Framework Project. However, in both cases, the level of detail provided in OPF is too broad for our purposes. For instance, the OPF reengineering phase description includes aspects such as management, quality, and testing; but does not include all the basic activities that we have identified in reengineering methods. Because of that, we decided to use another approach for defining our reengineering framework, but we still using OPF for assessing the analysis of existing reengineering methods, as a kind of classification schema. On the other hand, *method chunks* are specific of the method reengineered and, so, its granularity level is too detailed for being part of the generic framework. However, we can observer that it is possible to abstract and generalize the concepts of the specific *method chunks* into a set of generic *method chunks*. There are several approaches on how to document, store and reuse the different method parts [4], [5], [23], [25], [27], [28] that could be used to define and customize ReeF. However, as we use *method chunks* during the definition of the method, we keep on using them for illustrating its customization, as it is done in [23], [27], [28]. Consequently, we assume that *method chunks* are stored in a *method base*.

Taking those aspects into account, we have adopted a research method that, given an existing reengineering method, abstracts and generalizes its *method chunks*. As a result we have ReeF, a generic reengineering framework, which can be further customized by using other *method chunks* previously stored in the *method base*.

In order to abstract the initial set of method chunks using method Reengineering, we analyse PR*i*M, a Process Reengineering *i** Method [13]. We consider this method adequate as starting point because, as detailed in Section 3: 1) it is constructed after a rigorous state of the art of business process reengineering techniques; 2) it makes use of widespread techniques and artefacts in its definition instead of proposing ad-hoc ones; 3) some of the underlying ideas are applicable to contexts other than business process reengineering; 4) as authors, we have experience in applying the method and, so, access to all the components that we want to abstract onto the customizable framework which is an information sometimes difficult to obtain whilst analysing other methods.

The definition of ReeF is done in two steps: abstraction and generalization. Abstraction is the process of extracting common features from specific examples, whereas generalization is the process of formulating general concepts by abstracting common properties of instances. During the abstraction process, the phases of PR*i*M are analysed in order to synthesize its *method chunks*, following the principles given in [28]. PR*i*M is a method specific for the process reengineering domain. Thus, for obtaining a generic framework, we need to apply a generalization process over other reengineering methods from different domains. As a result, a new set of *method chunks* is obtained, with the particularity that the method artefacts (namely, the techniques, modelling languages, tool support and roles involved) are specified by stating its generic definitions instead of their particular ones. Also, special emphasis is given on the generic intention (the goal) that each *method chunk* pursues. The generic framework is then defined by analysing and reconciling all the obtained elements.

The customization of ReeF is done by applying the following steps: refinement, operationalization and combination. During refinement, the generic definitions stated in the *method chunks* of ReeF, are refined into specific ones for the domain of application. During the operationalization step, the refined statements of ReeF are used for selecting from the *method base* those *method chunks* that better accomplish a certain purpose. In order to facilitate this step, the *method chunks* can be classified according to a set of criteria [23], [27]. Finally, during combination, the selected method chunks are combined in order to obtain the new method. As we have mentioned, these steps can also be done by using other methods [5], [25].

Fig.1 presents an overview of the research method. We observe that the validation of ReeF is twofold. On the one hand, the proposed research method used for the definition of ReeF ensures that the different reengineering methods analysed can be successfully defined as instances of the framework. On the other hand, we define a new method for the domain of software architectures with the objective of validating its customization. The new method, called SAR*i*M, is then defined by customizing

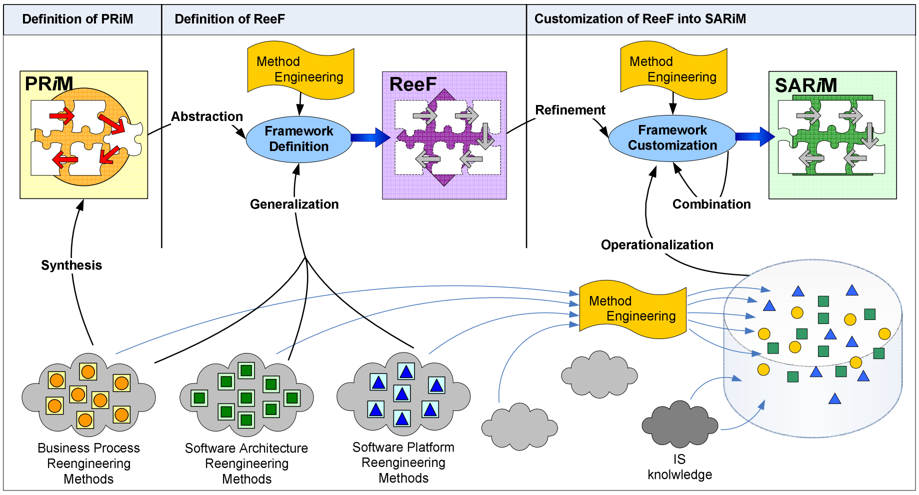


Fig. 1. Overview of the Research Method used for defining ReeF

ReeF, and combines *method chunks* from existing reengineering methods with specific techniques from the software architectures domain.

3 PR*i*M: A Process Reengineering *i** Method

In our previous research we defined PR*i*M [13], a Process Reengineering *i** Method that addresses the specification of Information Systems from the process reengineering perspective. The *i** framework [31] is a well consolidated goal-oriented approach that allows to model Information Systems in a graphical way, in terms of actors and dependencies among them. The use of the *i** framework in this context provides an appropriate milieu where the current process rationale is modelled by means of intentional concepts and the evaluation of the alternatives is done by analyzing the rationale behind the modelled intentional concepts.

Analysis and evaluation of *i** models is commonly done in a qualitative manner by using the analysis capabilities provided by the Strategic Rational Model. Instead, a goal of PR*i*M is to address the evaluation of alternatives from a quantitative point of view by applying structural metrics over the *i** models as proposed in [10], [11]. According to [9], one of the problems of the *i** framework is the repeatability when constructing the models. As repeatability is a fundamental property when applying structural metrics and it is not ensured by other *i** modelling techniques [14], the main motivation behind PR*i*M definition has been to ensure this property. Because of that, during the definition of PR*i*M we analysed several well-known business process reengineering and requirements engineering methods [12] in order to incorporate in the new method the adequate techniques, roles and artefacts. We highlight these included elements in the description of the method provided below, and summarize them in Table 1. We also remark that PR*i*M is defined upon the business process reengineering phases presented in [31] but adding a first preliminary phase for obtaining the information of the current processes.

The first phase of PR*i*M involves capturing and recording the information about the current process in order to inform further phases. The approach adopted is based on the RESCUE method [19] and, as a result, requirements engineers produce Human Activity Models (hereafter, HAM). During the second phase, the *i** model is build. In order to ensure repeatability when constructing the models, PR*i*M provides concrete guidelines that transform the information in Detailed Interaction Scripts (hereafter, DIS) to *i** elements. Thus, one of the activities the first phase is to adapt the information on the HAM to DIS. As both approaches share a common structure, simple transformation rules are provided in order to do it, and consistency checks are defined latter on the method for checking that they have been correctly applied.

In the second phase, the *i** model is built in two differentiated steps in order to distinguish the functionality performed by the stakeholders from their strategic intentionality. This approach is based on the semantically distinction of descriptive goals and prescriptive goals given in [2]. Therefore descriptive goals are modelled on the *operational i* model* by using the information in the DIS, and prescriptive goals are modelled on the *intentional i* model*. As a result of this process a *complete i* model* of the current process is obtained.

Table 1. Phases of PR*i*M, detailing the techniques, activities, inputs and outputs involved

Phase	Activity	Input	Techniques	Roles	Output
Phase 1: Analysis of the current process					
	Analysis of the current process	Current process	Observation	Process analyst	Human Activity Diagrams (HAM)
Phase 2: Construction of the *i model of the current process**					
	Transformation	HAM	Transformation rules	*i** modeller	DIS
	Actor Identification and modelling	DIS	Analysis of HAM	*i** modeller	*i** model actors
	Building the Operational *i** model	DIS	Transformation Rules	*i** modeller	Operational *i** model
	Building the Intentional *i** model	Operational *i** model	Provided Guidelines	Process analyst	Intentional *i** model
	Checking the Complete *i** model	Intentional *i** model	Consistency checks	*i** modeller	Complete *i** model
Phase 3: Generation of alternatives for the new process					
	Reengineering the current process	Complete *i** model	Requirements Elicitation Patterns	Requirements engineer	Enriched *i** model
	Adding new actors to the process	Enriched *i** model	Analysis of the market	Process designer	Actors for an *i** alternative (one)
	Reallocating responsibilities	Enriched *i** model, Actors	Provided Guidelines	Process designer	Alternative *i** model (one)
	Checking the consistency	Alternative *i** models (all)	Consistency Checks	*i** modeller	Consistent *i** alternatives
Phase 4: Evaluation of alternatives for the new process					
	Choosing suitable properties	Extended *i** model	Observation of needs from model	Process analyst	Properties
	Defining property metrics	Properties	Definition guidelines	Process analyst	Property metrics
	Evaluating alternative models	Consistent *i** alt. Metrics	Evaluation principles	*i** modeller	Evaluation Results
	Evaluation Trade-off analysis	Evaluation results	Trade-off analysis	Process analyst	Suitable *i** model solution
Phase 5: Specification of the new Information System					
	Specification of the new IS.	Suitable *i** model solution	Transformation guidelines	*i** and Use Case modellers	Use Case model of new IS.

The first activity of the third phase is to obtain the goals of the new process, which is done by using the *complete i* model* of the current process and applying KAOS [8] for analysing it. As KAOS and *i** are both goal-oriented, the acquired goals are added to the *complete i* model*, yielding to the *enriched i* model*. With the aim of satisfying these goals, several process alternatives are systematically generated by adding new *i** actors (which are mainly human, software or hardware), removing some of the existing ones and reallocating the responsibilities between them. This process is guided by the aim to satisfy the different new goals on the *enriched i* model*, which is done by applying the techniques proposed in [20]. As a result, several *alternative i* models* are produced.

In the fourth phase, the different *alternative i* models* are evaluated by applying structural metrics over them [10], [11]. Trade-off analysis is needed in order to select the most suitable solution. Finally, in the fifth phase, PR*i*M proposes the generation of the new Information System specification from the *i** model of the chosen alternative which follows the work proposed by [29].

The PR*i*M method is based on an exhaustive state-of-the-art on business process reengineering methods [12] complemented with well established requirements engineering techniques such as KAOS [8]. The use of these techniques provides an additional strength to all the phases defined on the method, and they have facilitated the development of J-PR*i*M [15], a tool that supports the application of the method. These are arguments that support using PR*i*M as starting point for formulating the framework.

Also we would like to remark the benefits of the use of *i** in PR*i*M. On the one hand, *i** supports all the phases of the method, allowing an *assembly of methods by association* [23], because no connection between the product models has to be done when combining the different *method chunks*. Actually, this also facilitates the substitution of most of the techniques applied on the phases for other *i** techniques with the same aims, without great modifications and without altering the result (e.g., the generation of alternatives can be done by using the organizational patterns proposed in [22]). On the other hand, as *i** is goal-oriented and agent-oriented, it allows reasoning at the goal and agent levels, which aligns with the strategic nature of reengineering processes. Consequently, in the *assembly of methods by integration* [23], goal-oriented and agent-oriented *method chunks* are easily adapted to represent the concepts in a unique *i** model (e.g., in phase 3, KAOS goals are represented in the *i** model).

4 Defining ReeF, a Customizable Reengineering Framework

In this section we explain the construction of ReeF in two differentiated processes, abstraction and generalization, starting from the PR*i*M method.

4.1 The Abstraction Process

In the Abstraction process we extract common reengineering features from the specific method PR*i*M. Thus, we use the Approach for Method Reengineering proposed in [28] over PR*i*M for achieving the proposed four main intentions: Define a section, Define a guideline, Identify a *method chunk*, and Define a *method chunk*. Due to the lack of space, we present directly the application of the method in our context; a complete description of the foundations can be found in [27], [28].

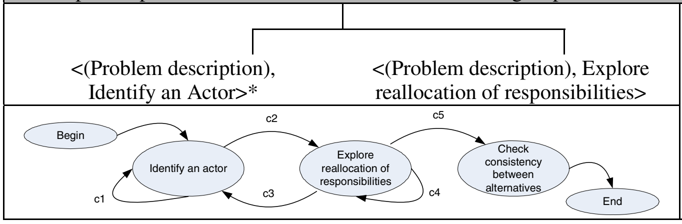
The PR*i*M method has a well defined process model and, so, in order to identify its sections we use the *functional strategy* in order to establish the *method map sections* from its phases. The intentions (or goals) of each phase of PR*i*M are identified and documented using the Method Reengineering suggested notation, as follows: Analyse the current process using Human Activity Modelling; Conceptualize the current process into an *i** model, Elicit requirements for the new process and explore different process alternatives based on them; Assess the generated process alternatives using evaluation techniques; and Create the specification of the new Information System.

When reviewing the guidelines associated to these intentions, we realize that the section “Elicit requirements for the new process and explore different process alternatives based on them” contains two different products that could be treated independently. Thus, we apply the *progression discovery strategy* and, as a result, the section is divided into two different ones: “Elicit requirements for the new process using a goal-oriented approach” and “Explore new process alternatives using process generation heuristics”. Once the sections are defined, the guidelines indicating how to proceed to achieve the objective of each identified section are also defined by applying Method Reengineering. For instance, the *method chunk* “Explore new process alternatives using process generation heuristics” has the strategic guideline that it is shown at the bottom of Fig. 2.

The *method chunks* are identified by using a *section-based discovery strategy*. We consider that each of the identified sections represents a *method chunk* because they can be reused separately outside its original method. Actually, as PR*i*M does so, we do not consider to apply any other strategy to identify more *method chunks*. Therefore we may define them already. At the top of Fig. 2 we present the descriptor for the *method chunk* “Explore new process alternatives using process generation heuristics”.

Situation:	**Application Domain:** Information systems, Business process reengineering **Design Activity:** Discover system requirements
Intention:	Explore new process alternatives using process generation heuristics
Origin:	PR*i*M method
Objective:	To help the process designer to explore different process candidate actors and generate the process alternatives that takes into account these actors.
Type:	Atomic
Aggregates:	<(Problem description), Explore a process alternative solution modelled in *i**>

<(Problem description),
Explore a process alternative solution with reallocating responsibilities between actors strategy>



<(Problem description), Identify an Actor>*
<(Problem description), Explore reallocation of responsibilities>
Begin
Identify an actor
Explore reallocation of responsibilities
Check consistency between alternatives
End
c1
c2
c3
c4
c5

c1: NOT all actors have been identified
c2: all actors have been identified
c3: an alternative has been generated
c4: NOT all the responsibilities have been reallocated
c5: all the alternatives have been generated.

Fig. 2. *Method chunk* “Explore new process alternatives using generation heuristics”

Once all the PR*i*M *method chunks* are identified, we abstract their intentions and the method artefacts used and, as a result, we obtain a set of *abstract method chunks*. Table 2 shows the results of these abstractions, where we can observe that the intentions of the PR*i*M *method chunks* are written in an abstract manner in order to help further customization of the method. This is done by substituting the PR*i*M specific artefacts (techniques, modelling languages, tool support and roles) for its equivalent generic artefacts, which are written between the symbols <>. The flow of the artefacts involved in the *abstracted method chunks* shows that they are treated in a specific order, hence establishing that they are sequential. In the fourth *abstracted method chunk* of Table 2, we show how the abstracted method artefacts are documented by stating a description and some of the examples of the analysis techniques, modelling languages, tool support and roles involved. The rest of the method artefacts abstraction is straightforward. A more formal documentation of the framework could be stated by using [6]. A complete catalogue of method artefacts can be found at the OPF repository [25].

Table 2. Generic notation for the intentions of the *abstracted method chunks* (abridged as *amc*)

amc 1:	**Analyse** [source] <domain artefact> **using** <analysis techniques> **obtaining** <analysed artefact>
amc 2:	**Conceptualize** <analysed artefact> **into** <model artefact>
amc 3:	**Elicit** <requirements artefact> **for the** [final] <domain artefact> **using** <elicitation techniques>
amc 4:	**Explore** [candidate] <domain artefact> **using** <generation techniques> **obtaining** [generated] <domain artefact>
	Techniques: Techniques and heuristics used to explore candidate solution artefacts (e.g., application of organizational patterns, application of architectural patterns, heuristics and guidelines for the generation of alternatives).
	Modelling language: Formalisms used to conceptualize the candidate solution artefacts (e.g., business process reengineering models, conceptual models, scenarios, architecture description languages, goal hierarchies, actor-dependency models such as *i**)
	Tool Support: Tools that aims at supporting the exploration of candidate solutions using an specific formalism (e.g., scenario generation tools, generation of alternative architectures tools)
	Roles Involved: Analyst, which is domain expert, responsible of exploring the solution artefacts (e.g., process analyst, software architectures analyst, systems analyst).
amc 5:	**Assess** [generated] <domain artefact> **using** <evaluation techniques>
amc 6:	**Create** [final]<specification artefact> **for the** [new] <domain artefact> **using** <model transformation techniques>

4.2 The Generalization Process

During the generalization process we formulate general concepts by analysing the common properties of other reengineering methods. Once the initial set of *method chunks* are identified, we apply again the Approach for Method Reengineering [28] to analyse more reengineering methods in order to obtain a generalization of the process. The undertaken review includes the methods used in the definition of PR*i*M (now studied from a Method Reengineering perspective) [2], [20]; business process reengineering methods [1], [24]; architecture reengineering methods [3], [21]; and platform reengineering methods [4], [32]. As a result, we obtain the *method chunks* of these processes. In Table 3 we present an excerpt of it by showing the intentions obtained from analysing the Scenario-based Software Architecture Reengineering method [3]. We observe that each intention corresponds to an *abstracted method chunk* with only one exception: after the elicitation of the functional requirements, the method assesses the current software architecture.

Table 3. Intentions proposed by the Scenario-based Software Architecture Reengineering [3]

Method	**Scenario-based Software Architecture Reengineering [3]**
amc 1:	These *method chunks* are not defined, as the method establishes as its input: the source <software architecture> conceptualized into <scenarios>
amc 2:	
amc 3:	**Elicit** <functional requirements> **for the final** <software architecture>
amc 5:	**Assess <**current software architecture**> using <**scenario-based evaluation**>**
amc 4:	**Explore** candidate <software architecture> **using** <QA-optimizing architecture transformations>
amc 5:	**Assess** generated <software architecture> **using** <scenario-based evaluation>
amc 6:	This *method chunk* is not defined. The output of the method is: <improved architecture design>

When analysing the *method chunks* obtained from applying Method Reengineering over all the previously mentioned reengineering methods, we observe the following:

- The analysed *methods chunks* present intentions that can be considered equivalent to the *abstracted method chunks*. For instance, all the methods share the intention of "Explore new solution artefacts", although they propose different guidelines to satisfy it.
- Not all the analysed methods present a sequential instantiation of the *abstracted method chunks*, as most of them omit some intentions. We remark that usually the omitted phases are the ones at the beginning or at the end of the process. For instance, in [3], [7], [21], the first two intentions are not mentioned as they assume that the information of the current situation is already studied and modelled for their purposes, but they all generate and evaluate candidate software architectures.
- Some of the studied methods propose a preliminary evaluation of the modelled process before the elicitation of new requirements. For instance, [3], [24], [32].
- Some of the methods allow iteration between the phases, allowing eliciting new requirements, exploring new solutions and evaluating them several times before choosing the final solution [3], [4], [7], [24].
- All the analysed methods have the *abstracted method chunks* for exploring and assessing the solution artefacts. However, in some of the methods the assessment is implicit in the exploration of the solutions as if it were a cycle between both phases. For instance, in [4] and [21] the designer generates the solutions according to its own criteria, which means an implicit evaluation of the current solution.
- All the studied methods have their intentions executed in the sequential order established by the *abstracted method chunks*. An extreme example of this is the work proposed in [24] where different reengineering processes can be generated from applying a set of map strategies, and the generated methods are compliant with ReeF.
- The method artefacts obtained in the studied *method chunks* are equivalent to those abstracted in ReeF and, although the proposed techniques come from different domains, their intentions and roles are an instance of the ones abstracted.
- All the methods use a modelling language for communicating between its phases. The common modelling languages are visual models (e.g., Use Case Maps [7], enterprise business process models [24]) and structured text (e.g., scenarios [3]).

Taking those considerations into account, we generalize the *abstracted method chunks* obtained in ReeF and we establish the following restrictions:

- There is a sequential order within the different *abstracted method chunks*, but it is possible to omit the ones at the beginning or at the end, as some methods do.
- It is possible to assess the source artefact after it is modelled, in order to inform the elicitation of requirements.
- It is possible to iterate between the phases: the evaluation of alternatives can inform a new elicitation of requirements; new alternatives are generated and evaluated; and so on and so forth, until a final solution is found.

As a result, the ReeF framework is composed by six phases, which are shown in Fig. 3. The blue arrows show the sequence of execution of the phases according to the *abstracted method chunks* allowing the diversions and iterations previously mentioned. The framework defines, for each of these phases, the work products needed (inputs) and produced (outputs) during the phases, the techniques (including the activities for obtaining the work products, the transformations between models and the tool support used) and the roles that are involved. As the framework is generic, customization has to be applied in order to instantiated it. We remark that during customization it is possible to define the new method by using different techniques for each of its iterations if needed.

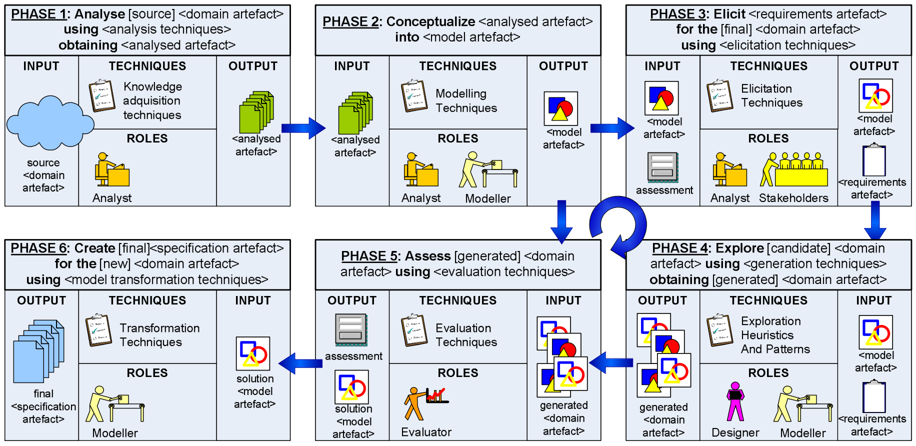

PHASE 1: Analyse [source] <domain artefact> using <analysis techniques> obtaining <analysed artefact>
INPUT
TECHNIQUES
OUTPUT
Knowledge adquisition techniques
ROLES
source <domain artefact>
Analyst
<analysed artefact>
PHASE 2: Conceptualize <analysed artefact> into <model artefact>
Modelling Techniques
Modeller
<model artefact>
PHASE 3: Elicit <requirements artefact> for the [final] <domain artefact> using <elicitation techniques>
Elicitation Techniques
assessment
Stakeholders
<requirements artefact>
PHASE 4: Explore [candidate] <domain artefact> using <generation techniques> obtaining [generated] <domain artefact>
Exploration Heuristics And Patterns
generated <domain artefact>
Designer
PHASE 5: Assess [generated] <domain artefact> using <evaluation techniques>
Evaluation Techniques
solution <model artefact>
Evaluator
PHASE 6: Create [final]<specification artefact> for the [new] <domain artefact> using <model transformation techniques>
Transformation Techniques
final <specification artefact>

Fig. 3. Phases, inputs, outputs, techniques and roles abstracted in ReeF

5 Customizing ReeF into SAR*i*M

As an example of application of the framework we propose the definition of SAR*i*M, a Software Architecture Reengineering *i** Method. The aim of SAR*i*M is to adapt the experience in using PR*i*M to the domain of software architectures. The use of *i** as a modelling language has several advantages. On the one hand, *i** allows to represent functional and non-functional requirements as well as business goals at the same level, thus bridging the gap that is usually found between requirements and architectures [16]. On the other hand, *i** has already been successfully used for the representation of software architectures [18]. As a result, the customization strategy followed in the SAR*i*M case has prioritized operationalization over refinement and combination (see Fig. 1).

Refinement. The generic intentions (or goals) defined in the *abstract method chunks* of ReeF are refined for the particular domain of software architecture in order to establish the main objectives to be satisfied in the new method, see Table 4. We observe that only the desired artefacts are refined and that the precise technique may still be undefined. As we do in PR*i*M, we do not consider the evaluation of the current software architecture before the elicitation of the new requirements.

Operationalization. Once the intentions are defined, we search into the *method base* those *method chunks* that better accomplish the intention. We propose to classify the *method chunks* in the database according to three dimensions: intention they support, domain they are designed for, and modelling language used. The reason for taking considering the modelling language is that, if the *method chunks* do not share the same modelling language, a transformation technique has to be applied between them, so it is recommended to take this aspect into account in order to facilitate further steps. However, other classification criteria for the *method base* can be used [3], [23].

Based on the refined intentions, the search for the appropriate *method chunks* in the *method base* is facilitated, as the set of candidate elements is delimited. We remark that the *method base* is not complete and not all the *method chunks* required may be found there. If this is the case, a study of other suitable methods has to be done and the resulting *method chunks* have to be added into the *method base*. This study may include reengineering methods but also well-know requirements engineering methods or guidelines for the application of patterns that, although not being defined as reengineering methods, may support some of the proposed phases.

In the third column of Table 4 we show whether the *method chunks* available in the method base are supported or not by PR*i*M. In the example presented in Table 3 we show the intentions of the *method chunks* for the Scenario-based Software Architecture Reengineering Method [3]. There, the fourth *method chunk* is scenario-based and proposed a set of architecture transformation guidelines based on quality attributes. As this intention satisfies the one we have refined in SAR*i*M, we use it.

The other phases that are not supported by PR*i*M are the analysis of the current software architecture and the elicitation of requirements for the future one. As there are no *method chunks* in the method base to support those phases, we analyze other methods for doing it. More precisely, we have searched in the field of requirements engineering and we have selected the Architecture Reconstruction Method [17] for the recovery and analysis of the current architecture, and the CBSP method [16] to be adapted to the *i** notation for the elicitation of the new requirements.

Combination. Once the *method chunks* are selected, method engineering techniques for assembling can be applied [5], [23], [25], [27], [28] in order to obtain the final method. The combination of the *method chunks* is out of the scope of this work, as it

Table 4. Refinement step, customazing ReeF in the domain of Software Architectures

Generic Intention in ReeF	Refinement into SAR*i*M	*Method chunks* Operationalization
Analyse [source] <domain artefact> **using** <analysis techniques> **obtaining** <analysed artefact>	**Analyze source** software architecture **using** *<architecture analysis technique>*	Not supported by PR*i*M: operationalized by the Architecture Reconstruction Method.
Conceptualize <analysed artefact> **into** <model artefact>	**Conceptualize** the software architecture **into** an *i** model	Supported by PR*i*M: needs previous transformation of the results into DIS
Elicit <requirements artefact> **for the** [final] <domain artefact> **using** <elicitation techniques>	**Elicit** quality requirements **for the final** software architecture **using** *<elicitation technique>*	Not supported by PR*i*M: operationalized by the CBSP method.
Explore [candidate] <domain artefact> **using** <generation techniques>	**Explore candidate** software architectures **using** *<generation techniques>*	Not supported by PR*i*M: use of the Scenario-based Software Architecture Reengineering Method.
Assess [generated] <domain artefact> **using** <evaluation techniques>	**Assess generated** software architecture **using** *i** structural evaluation techniques.	Supported by PR*i*M: needs the generated architectures to be represented as *i** models.
Create [final]<specification artefact> **for the** [new] <domain artefact> **using** <model transformation techniques>	**Create final** specification **for the new** software architecture **using** *i** to use cases transformation techniques.	Supported by PR*i*M: can be applied directly from the previous phase.

has already been addressed in [23], [27]. We just remark that, following the criteria in [23] all the *method chunks* are combined following the established order and using the *assembly by association*, where transformation techniques are applied in order to transform *i** models to scenarios. In the *method chunks* for requirements elicitation and architectures generation, we apply an *assembly by integration*, as the tight link between *i** and requirements engineering techniques, facilitates it.

6 Conclusions and Future Work

In this paper we have argued that the evolution of Information Systems very often leads to a reengineering activity. There are a lot of methods proposed in the literature at different levels (business processes, software processes, software architectures, etc.). This methods support reengineering both consciously, by applying the term reengineering in its proposal; and unconsciously, by mentioning the phases that characterize reengineering. However, as far as we know, there is not a common framework to reason about reengineering and this has been the motivation of our proposal. ReeF has been defined following the principles of Method Engineering because this technique is specially well-suited when defining new methods based on existing ones. As a result, the advantage of applying the framework is twofold:

- It provides a common umbrella under which the different existing reengineering proposals may be analysed, compared for possible adoption, customized to particular contexts and even composed to deal with reengineering at different levels. In particular, an existing method could be enlarged to deal with some activity not covered in its definition, or some technique may be changed with some other identified as similar.
- It allows formulating new reengineering approaches starting from that framework, not only facilitating that task, but also providing an ontology of reference and the possibility of reusing methods, techniques, models and tools from a common experience base.

ReeF is not intended to deliver an exhaustive catalogue with all the possible phases and techniques, but instead it serves as a generic, customizable framework, which provides, among other things, different levels of abstraction and the possibility of choosing between different characteristics. More precisely, we argue that the framework satisfies the following guiding principles proposed by the OPF [25]:

- **Flexibility.** In order to allow maximum flexibility when customizing, the phases of ReeF provide: atomicity, in the way that the activities it proposes are related to only one concept of the reengineering activities; optionality, certain phases can be avoided if the customization requires so; and iteration, in those methods that require several iterations of some of the phases.
- **Standardization.** Reef uses the most common terminology in the business process reengineering field. For techniques, roles and activities it uses the already standardized terminology and concepts coming from the OPF.
- **Completeness.** ReeF is complete in the sense that it includes all the elements that may be needed in a reengineering process. Although it not provides a complete

repository of elements for instantiate the framework, it provides techniques for constructing this repository.

- **Openness.** ReeF remains open in the sense that there is not a closed list of elements and also because it is not necessary to instantiate all those elements, allowing the method engineer to customize them accordingly to its goals.
- **Reengineering Best Practices.** ReeF is based on the abstraction and generalization of well-know reengineering methods and related requirements engineering techniques.
- **Usability.** ReeF facilitates usability by providing guidelines for its customization, as it is shown in the customization of ReeF into SAR*i*M.
- **Reuse.** The framework supports reuse of methods providing the context where to customize the method and a set of elements as examples.

Further work will involve the application of ReeF on the combination of reengineering methods that work in different domains (e.g., business process reengineering and architecture reengineering). This includes the definition of more *method chunks* and method artefacts into the *method base* and how to document and classify them in order to facilitate their customization. We are mainly interested in the use of *i** and the *method chunks* proposed in PR*i*M as a basis for this process and we want to adapt J-PR*i*M [15] in order to provide tool support for the whole process.

Acknowledgements

This work has been partially supported by the CICYT programme, project TIN2004-07461-C02-01. Gemma Grau work is supported by an UPC research scholarship.

References

1. van der Aalst, W.M.P., van Hee, K.M.: Framework for Business Process Redesign. In: Proceedings of the Fourth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 36–45 (1995)
2. Antón, A.I., McCracken, W.M., Potts, C.: Goal Decomposition and Scenario Analysis in Business Process Reengineering. In: Proceedings of CAiSE 1994. LNCS, vol. 811, pp. 94–104. Springer, Heidelberg (1994)
3. Bengtsson, P., Bosch, J.: Scenario-based Software Architecture Reengineering. In: Proceedings of the 5th International Conference on Software Reuse, pp. 308–317 (1998)
4. Bouillon, L., Vanderdonckt, J., Chow, K.C.: Flexible Re-engineering of Web Sites. In: Proceedings of the 9th International Conference on Intelligent user interface (2004)
5. Brinkkemper, S., Saeki, M., Harmsen, F.: Assembly Techniques for Method Engineering. In: Proceedings of CAiSE 1998. LNCS, vol. 1413, pp. 381–400. Springer, Heidelberg (1998)
6. Brinkkemper, S., Saeki, M., Harmsen, F.: A Method Engineering Language for the Description of Systems Development Methods. In: Proceedings of CAiSE 2001. LNCS, vol. 2068, pp. 473–476. Springer, Heidelberg (2001)
7. de Bruin, H., van Vliet, H.: Scenario-based Generation and Evaluation of Sofware Architectures. In: Proceedings of the Third International Conference on Generative and Component-Based Software Engineering, LNCS, vol. 2186, pp. 128–139. Springer, Heidelberg (2001)

8. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed Requirements Acquisition. Science of Computer Programming 20(1-2), 3–50 (1993)
9. Estrada, H., Martínez, A., Rebollar, O., Pastor, J.: An Empirical Evaluation of the i* in a Model-Based Software Generation Environment. In: Proceedings of CAiSE 2006. LNCS, vol. 4001, pp. 513–527. Springer, Heidelberg (2006)
10. Franch, X.: On the Quantitative Analysis of Agent-Oriented Models. In: Proceedings of CAiSE 2006. LNCS, vol. 4001, pp. 495–509. Springer, Heidelberg (2006)
11. Franch, X., Grau, G., Quer, C.: A Framework for the Definition of Metrics for Actor-Dependency Models. In: Proceedings of RE 2004, pp. 348–349
12. Grau, G.: State of the Art for the Systematic Construction and Analysis of i* Models for assessing COTS-Based Systems Development. Research Report LSI-06-38-R. Available at: http://www.lsi.upc.edu/ techreps/files/R06-38.zip
13. Grau, G., Franch, X., Maiden, N.A.M.: A Goal Based Round-Trip Method for System Development. In: Proceedings of REFSQ 2005, pp. 71–86 (2005)
14. Grau, G., Cares, C., Franch, X., Navarrete, F.J.: A Comparative Analysis of i* Agent-Oriented Modelling Techniques. In: Proceedings of SEKE 2006, pp. 657–663 (2006)
15. Grau, G., Franch, X., Ávila, S.: J-PRiM: A Java Tool for a Process Reengineering i* Methodology. In: Proceedings of RE 2006, pp. 352–353 (2006)
16. Grünbacher, P., Egyed, A., Medvidovic, N.: Reconciling software requirements and architectures with intermediate models. Software and Systems Modeling 3(3), 235–253 (2004)
17. Guo, G.Y., Atlee, J.M., Kazman, R.: A Software Architecture Reconstruction Method. In: Proceedings of WICSA 1999, pp. 15–34 (1999)
18. The i* wiki at: http://istar.rwth-aachen.de/ Last Accessed : November 2006
19. Jones, S., Maiden, N.A.M.: RESCUE: An Integrated Method for Specifying Requirements for Complex Socio-Technical Systems. Book chapter in Requirements Engineering for Sociotechnical Systems, Idea Group Inc. (2004)
20. Katzenstein, G., Lerch, F.J.: Beneath the Surface of Organizational Processes: A Social Representation Framework for Business Process Redesign. ACM Transactions on Information Systems 18(4), 383–422 (October 2000)
21. Kim, M., Lee, J., Kang, K.C., Hong, Y., Bang, S.: Re-engineering Software Architecture of Home Service Robots: A Case Study. In: Proceedings of ICSE 2005, pp. 505–513 (2005)
22. Kolp, M., Giorgini, P., Mylopoulos, J.: Organizational Patterns for Early Requirements Analysis. In: Proceedings of CAiSE 2003. LNCS, vol. 2681, pp. 617–632. Springer, Heidelberg (2003)
23. Mirbel, I., Ralyté, J.: Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering, 11(1) (2005)
24. Nurcan, S., Rolland, C.: A multi-method for defining the organizational change. Information and Software Technology 45(2), 61–82 (February 2003)
25. The OPEN Process Framework (OPF) at: www.opfro.org. Last accessed: November 2006.
26. Pressman, R.S.: Software Engineering: a Practitioner's Approach. In: International Edition, 6th edn. McGraw-Hill, New York (2005)
27. Ralyté, J.: Ingénierie des méthodes par assemblage de composants. Thèse de doctorat en informatique de l'Université Paris 1 (Janvier 2001)
28. Ralyté, J., Rolland, C.: An Approach for Method Reengineering. In: ER 2001. LNCS, vol. 2224, pp. 471–484. Springer, Heidelberg (2001)
29. Santander, V.F.A., Castro, J.F.B.: Deriving Use Cases from Organizational Modeling. In: Proceedings of RE 2002, pp. 32–39 (2002)

30. Smith, J.D., Hybertson, D.: Implementing Large-Scale COTS Reengineering within the United States Department of Defense. In:Proceedings of ICCBSS 2002. LNCS, vol. 2255, pp. 243–256. Springer, Heidelberg (2002)
31. Yu, E.: Modelling Strategic Relationships for Process Reengineering, PhD. thesis, University of Toronto (1995)
32. Zhang, W., Jarzabeg, S., Loughran, N., Rashid, A.: Reengineering a PC-based System into the Mobile Device Product Line. In: Proceedings of the 6th International Workshop on Principles of Software Evolution, pp. 149–160 (2003)