A Framework for Integrating Business Processes and Business Requirements*

Raman Kazhamiakin Marco Pistore
DIT, University of Trento
Via Sommarive 14, I-38050, Trento, Italy
{*raman,pistore*}*@dit.unitn.it*

Marco Roveri
ITC-irst
Via Sommarive 18, I-38050, Trento, Italy
roveri@irst.itc.it

Abstract

Service-oriented architectures and Web service infrastructure provide the ideal framework for interconnecting organizations and for defining distributed business applications. The possibility to exploit business process definition and execution languages is particularly relevant for capturing the process-oriented nature of these applications. However, business processes by themselves are not enough to manage the changes and to allow an organization to continuously adapt its business model to the typical needs of distributed applications. To achieve this flexibility, it is of uttermost importance to link the business processes to the organizational strategy and to the business goals that motivate the need of these processes.

In this paper we propose a framework for representing strategies and goals of an organization in terms of business requirements. The framework allows to describe how an organizational strategy is operationalized into activities and implemented by business processes. It also allows to represent the assumptions on the interactions between the different business applications. Finally, this framework allows for the usage of formal analysis techniques, in particular Model Checking, to pinpoint problems and to identify possible solutions in this domain.

1. Introduction

The growing use of information technology and the high level of dynamics and interoperability introduced by the Internet provide new ways for interconnecting enterprises and customers that where not possible in the past. In particular, Web service technology is rapidly emerging as one of the most practical approaches for the integration of customers', vendors', and business partners' applications. The number of individual Web services made available by companies is continuously growing, but the real value to an organization comes only when these companies are able to connect services together. This forces transaction and integration costs to be driven down and makes the traditional static hierarchical structure less attractive both from a strategic and from an economical perspectives. Organizations should be considered in terms of business processes rather than functions, integrating activities in cross-functional and cross-enterprise chains [16]. This integration is made more difficult by the fact that, while aiming to work together and to provide inter-enterprise business applications, usually companies do not want to uncover the internal structure of their business processes to each other. Moreover, they want to keep the ability to rethink and reorganize their business processes and to modify accordingly the implementation in order to adapt their strategy to changes and innovations.

Web services provide the basic technology for obtaining the loosely coupled integration required by business processes. Indeed business process description languages like BPEL4WS [1] provide means to describe services without necessarily revealing the internal structure of the processes. However, in order to be effectively used for composing business processes, the technological infrastructure provided by Web services needs to be enriched with tools and methodologies that support the development of this composition and that permit to manage the changes and adaptations of business models and of business processes. This introduces the necessity of determining the implication of the business strategy and goals changes in the software system. To this purpose, business goals and business requirements need to be made explicit in the business process model. A "strategic" description needs to be provided of the different actors in the business domain with their goals and with their mutual dependencies and expectations. Furthermore, a tight integration of the business requirements and of the business process specification is necessary. This permits requirements traceability, i.e., to see how the modifications of requirements or process specifications affect each other.

One of the major prerequisites for an effective process integration is reliability. The process definition that is obtained from the business requirements model should be con-

*This work has been supported in part by the FIRB-MIUR project RBNE0195K5 "Astro".

sistent with the requirements and with the goals it aims to achieve. And, whenever processes of different partners are composed into a new business application process, the composition should respect the goals and constraints of every participant. In other words, the business application should match the business requirements model. Automatic formal analysis and verification tools are necessary to verify this matching and to check that this match is maintained when requirements or processes change.

In this work, we describe a framework for the business process requirements modeling that we are currently defining. The design process starts from a description of the strategic goals and needs of an organization. These are refined and operationalized into tasks and activities, which are then transformed in turn into business processes and Web services. Formal annotations are used at all levels to define constraints to business requirements and to business process models. For modeling business requirements we exploit the Tropos software development methodology [3]. Tropos provides the notations to capture the business requirements of the participants in the domain, their mutual dependencies and expectations. We also exploit the formal counterpart of Tropos, the Formal Tropos language [9], that provides rich notations for the definition of constraints and properties of the modeled domain, and that is amenable for formal verification. Tropos and Formal Tropos are extended to target the specific aspects of Web services technology. In particular, to deal with business processes, BPEL4WS specifications are linked to the requirements models.

The paper is organized as follows. In Section 2 we describe the business process modeling capabilities offered by Web services and discuss the necessity to explicitly introduce requirements and to integrate them with the processes. In Section 3 we show how the business process requirements can be modeled in Tropos and how these requirements specifications may be presented formally. Section 4 describes the way the requirements models are integrated with business process models and specifications. Formal analysis and verification of the requirements and process models is described in Section 5. We conclude the paper with a briefly review of related works and with some remarks on future work.

2. Business Processes and Business Requirements

2.1. Business Processes

The description of the structure and behavior of the different business activities within an organization has been deeply investigated, for instance in the framework of business process modeling. However, the integration of business processes that are distributed among different organizations is still a challenging problem.

Web services are an emerging technology for building complex distributed systems focusing on interoperability, support for efficient integration of distributed processes, and uniform representation of applications. They allow companies to describe the external structure of their processes and how they can be invoked and composed. Web service support the interactions among the different partners by providing a model of synchronous or asynchronous exchange of messages. These exchanges of messages can be composed into longer business interactions by defining protocols constraining the behavior of all partners.

The terms orchestration and choreography are often used to refer to the two key aspects of process composition [14]. In *orchestration*, the composition is considered from the perspective of one of the business parties. The focus is on the interaction that the business process under consideration performs with internal and external Web services in order to carry out its task. Orchestration is usually private to the business party, since it contains reserved information on the specific way a given process is carried out. *Choreography*, on the other hand, describes the interactions for a global, neutral perspective, in terms of valid conversations or protocols among the different parties. Choreography is usually public, since it defined the common rules that define a valid composition of the distributed business processes in the business domain.

Web services have developed different languages for orchestration and choreography (BPEL4WS, WSFL, WSCI...). Among them, BPEL4WS [1] is quickly emerging as the language of choice for the description of process interactions. BPEL4WS provides core concepts for the definition of business process in an implementation-independent way. It allows both for the definition of internal business processes and for describing and publishing the external business protocol that defines the behavior of the interaction. Therefore, BPEL4WS permits to describe both the orchestration and the choreography of a business domain with an uniform set of concepts and notations.

2.2. Business Requirements

While developing distributed business services, the designers usually focus on the "how", that is on the way a business process is implemented by means of standard business process description languages. However, these languages are not able to describe the business goals and strategies of an organization, its expectations over external services, and the links existing between these goals and expectations and the corresponding business processes.

By business requirements we mean all those aspects of the description of business process that are related to the

strategy and the rationale of on organization (the "why" and the "what"), and that precede and motivate the definition of specific processes (the "how").

Similarly to what happens for business processes, also in business requirements we can distinguish an orchestration and a choreography perspective. In the *orchestration* perspective, the point of view of a particular business party is taken, its strategic goals are described, the definition of the business processes needed to achieve these goals is made evident along with the decision of what services to implement internally and what external services to exploit. The requirements from the *choreography* perspective, on the other hand, aim to describe the interaction opportunities that are present in the business domains. These opportunities are defined in terms of possible matches between the services required by certain actors and the services provided by other actors. Along with these intent/offer matches, the choreographical business requirements also define the business rules of the domain, namely the shared assumptions and constraints on the correct interactions inside the business domains. Orchestration is supposed to be private to the party, as it contains information on the business strategy that the party may not want to disclose to external organizations, while choreography is supposed to be publicly available to all participants. Correct business processes of a party should respect goals and constraints both of its own orchestration and of the shared choreography of the business domain.

2.3. A Case-Study

We consider a case-study in the field of public welfare, extracted from a larger domain concerning the local government of Trentino (Italy). In the case-study we consider a senior citizen that aims at being assisted, e.g., receiving services like transportation or meals at home. The assistance to citizens is provided by a Health-care Agency, that is run by the Local Government, and that aims at providing fair assistance to citizens. The Health-care Agency depends on external providers for the actual delivery of the requested services. The financial aspects of the Local Government are handled by a Bank that is in charge of paying the service providers and of asking the citizen to cover part of the costs of the used services. The interaction among the different parties is required to happen via Web services. The business goal of the citizen consists in receiving assistance, while the Health-care Agency is willing to provide assistance only to citizen that satisfy given eligibility criteria.

3. Business Requirements Modeling in Tropos

Tropos is a goal-driven, agent-oriented software development methodology that aims to cover all the phases of the development process starting from early requirements [3]. It is founded on the premise that during early requirements it is important to understand and model the strategic aspects underlying the organizational setting within which the software system will eventually function. By understanding these strategic aspects one can better identify the motivations for the software system and the role that it will play inside the organizational setting. The methodology allows for the incremental refinement of the strategic models via goal/task decomposition and operationalization both at the informal and formal level.

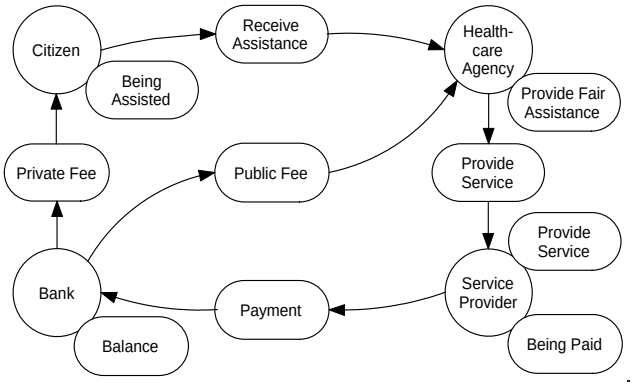


Figure 1. High level requirements model.

In this paper we show that (a suitable extension of) Tropos can be used to define business requirements and to integrate them with the corresponding business processes. We will use the health-care assistance case-study to illustrate the approach.

3.1. Tropos for Web Services

Figure 1 is a Tropos diagram that describe the *actors* (circles) participating to the case-study, and their strategic high-level *goal*s (the ovals attached to the actors). For instance, in the diagram we have the Citizen that aims at being assisted; the HealthcareAgency that aims at providing a fair assistance to the citizens; the ServiceProvider which goal is to provide the requested services; and the Bank which handles the government's finances.

Tropos allows for the description of the interactions among the different parties of the domain at the strategic level relying on the intent/offer matching mechanism represented in the diagram by means of *dependencies* (the ovals linked to two different actors). For instance, the Citizen depends on the Health-care Agency for being assisted, and this is formulated in the model with dependency ReceiveAssistance from Citizen to HealthcareAgency. This diagram can be seen as a very high-level choreography representation of the requirements of our case-study.

Starting from this high-level view of the organizational

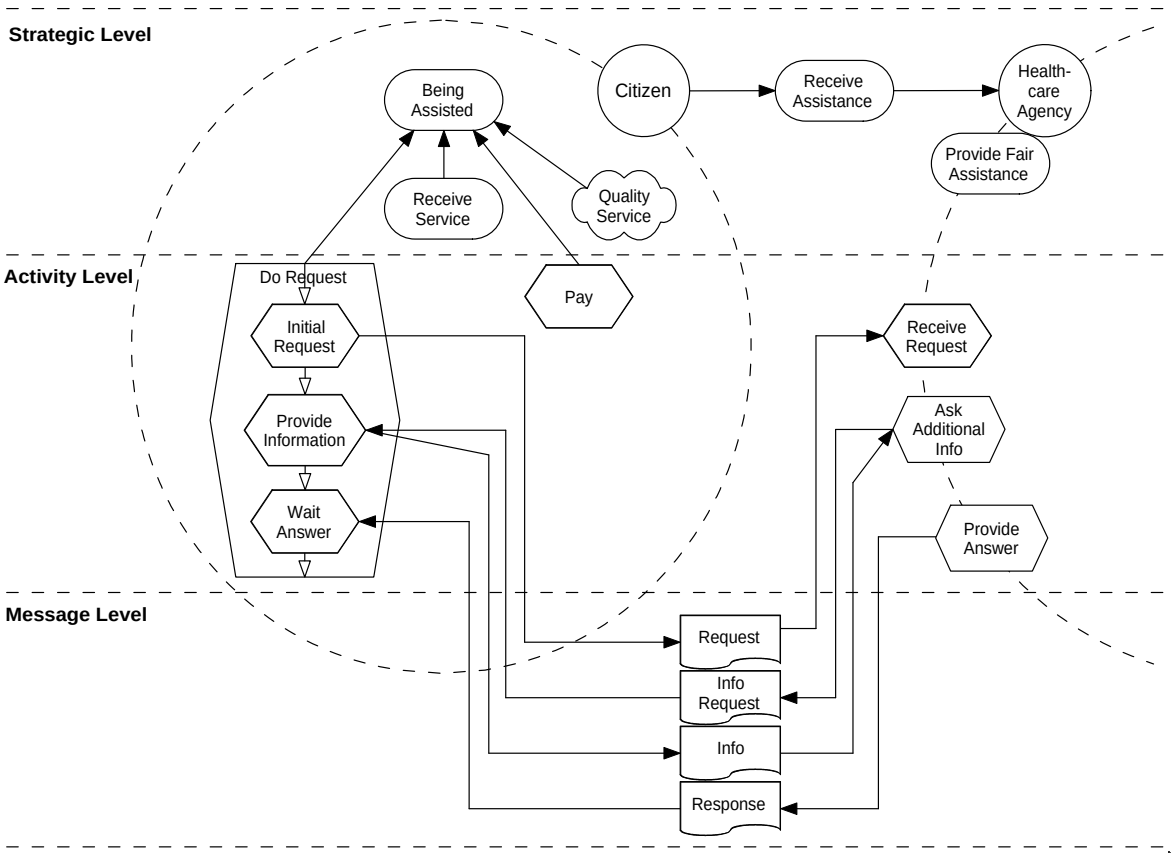


Figure 2. Requirements model refinement.

or business system, Tropos proceeds with an incremental refinement process (see Figure 2). This refinement starts with a goal analysis, where the high level goals of one of the actors are refined into sub-goals and eventually operationalized into tasks. In our case-study, the goal analysis is very simple: the Citizen refines the goal BeingAssisted into the two tasks (hexagons) DoRequest and Pay, the goal ReceiveService, and the soft-goal (cloud) QualityService. The tasks are supposed to be implemented by software modules, while the goals that remain in the model after the goal analysis represent activities that are not carried out electronically (e.g., the assistance services are physically delivered to the citizen). Finally, soft-goals are used to describe non-functional requirements, with no clear-cut criteria as to when they are achieved (e.g., the citizen has some requirements on the quality of the services delivered to him).

The goal analysis phase is followed by a task refinement phase, where the high-level tasks are decomposed into subtasks. In Figure 2, task DoRequest is further refined into InitialRequest, ProvideInformation, WaitAnswer. In this simple case, the three sub-tasks are composed sequentially. Other forms of task decomposition are also possible, corresponding to the other typical ways of combining activities in activity diagrams (parallel composition, choice, iteration...).

The task decomposition procedure ends once we have identified all basic tasks that define the business process. As a last step in the definition of business requirements, we associate to the basic tasks the messages that describe the basic interactions among actors. For instance, task InitialRequest requires to send a message Request to the HealthcareAgency. This message is received and processed by the task ReceiveRequest of the HealthcareAgency. The task AskAdditionalInfo of the HealthcareAgency is implemented by sending a message InfoRequest to the Citizen which receives and processes it within task ProvideInformation and responds with an Info message. Once sufficient information has been gathered, the HealthcareAgency sends a Response message to the Citizen.

The steps of the refinement are represented in the Figure 2 as three levels: a strategic level, an activity level and a message level. All these levels are part of the requirements model, in the sense that they define different aspects of the requirements a valid implementation is supposed to respect.

We remark that Figure 2 represents the point of view of the Citizen, therefore it can be seen as an “orchestration” requirements diagram. Clearly, the “internal” refinement

done for goal BeingAssisted has to take into account and to reflect the "contract" that governs the way the assistance is provided to the Citizen by the HealthcareAgency (and by the other actors of the diagram). Indeed, the diagram in Figure 2 shows also the links that exist between the Citizen and the HealthcareAgency, both at the goal level and at the message level. However, no description of the internal structuring of the HealthcareAgency is represented in the diagram, since this information is not available to the Citizen.

It is worth to be noticed that in this example we have adopted a top-down strategy for transforming high-level goals into interactions. Other strategies, e.g., a bottom-up approach from the messages to the goals, or a middle-out approach starting from the activities, are also possible in this framework.

3.2. Formal Business Requirements Specification

The Tropos methodology allows for extending the Tropos diagrams with formal annotations expressed in Formal Tropos (hereafter FT). The FT annotations specify the valid behaviors and the relations among the different actors, dependencies, goals, tasks and messages in the model. At the strategic level the FT annotations specify the conditions on goal creation and fulfillment, and assume/guarantee conditions on delegations. At the activity level, they define pre- and post-conditions on tasks and sub-tasks. Even more important, FT annotations allow to link these two levels and the underlying message level. The key advantage of FT with respect to other approaches is that it defines the dynamic aspects of a model and supports its formal verification already at the requirements level, without requiring an operationalization of the specification, e.g., into BPEL4WS processes. A precise definition of FT and of its semantics can be found in [9]. Here we present the most relevant aspects of the language based on the case-study. An excerpt of the FT annotations associated to the DoRequest task can be found in Figure 3.

FT gives a description of the different objects in the modeled domain, which is similar to a class declaration. In particular, a list of attributes is associated to each of these classes. Each attribute has a *sort* which can be either primitive (boolean, integer...) or can be a reference to other class. Some special attributes are associated to each kind of class in the FT specification. Goals and tasks are associated to the corresponding actor with the special attribute **Actor**. Similarly, **Depender** and **Dependee** attributes of dependencies represent the two parties involved in a delegation relationship. Attribute **Super** for goals and tasks denotes the parent goal or task.

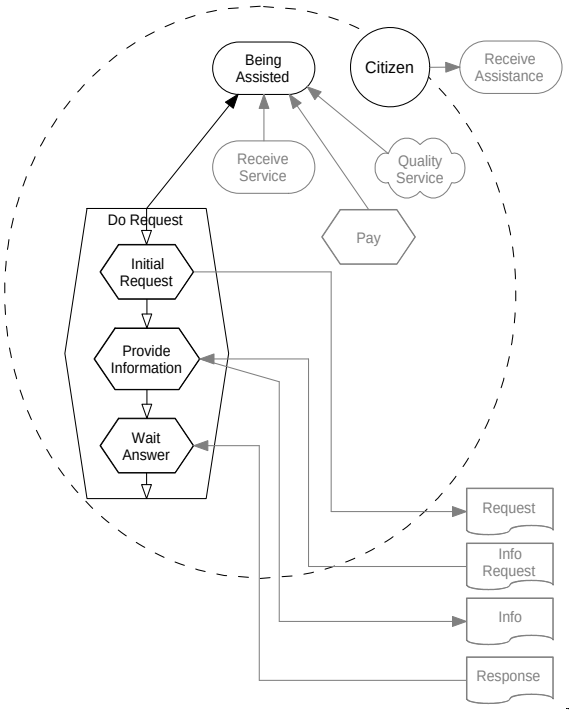
An important aspect of FT is its focus on the conditions for the *fulfillment* of goals and tasks. These are characterized by a **Mode**, which declares the modality of their fulfillment. The two most common modalities are **achieve** (which means that the actor expects to reach a state where, e.g., the goal has been fulfilled) and **maintain** (which means that the fulfillment condition has to be continuously maintained). For instance, dependency ReceiveAssistance is of type maintain, to capture the fact that this "contract" between citizen and health-care agency has to be maintained over time. On the other hand, task DoRequest (as most of the tasks) is of type achieve, since the citizen aims at reaching a state where this task is terminated.

Behavioral aspects of the objects in the model and relations among them are annotated in the FT specifications as constraints. They allow to capture the conditions on the goals creation and fulfillment and pre- and post-conditions on tasks and sub-tasks. **Creation** constraints define conditions that should be satisfied when a new instance of a class is created. In the case of goals and tasks, the creation is interpreted as the moment when the associated actor begins to desire the goal or to perform the task. **Fulfillment** constraints should hold whenever a goal is achieved or a task is completed. Creation and fulfillment constraints are further distinguished as sufficient conditions (keyword **trigger**), necessary conditions (keyword **condition**), and necessary and sufficient conditions (keyword **definition**).

In FT, constraints are described with formulas in a typed first-order linear-time temporal logic. Besides the standard boolean and relational operators, the logic provides the quantifiers $\forall$ and $\exists$, which range over all the instances of a given class, and a standard set of linear-time temporal operators. The latter include operator **X**, which defines a condition that has to hold in the next state of the evolution of the system, operator **F**, which defines a condition that has to hold eventually in the future, and operator **G**, which defines a condition that has to hold in all future states.

In the FT specification of Figure 3, the definitions of task DoRequest and its subtasks model the life-cycle of the activity. For instance, the fulfillment definition of ProvideInformation specifies that this task aims to send an information message in reply of every incoming information request and it is fulfilled if all information requests have been answered. The DoRequest task is considered to be fulfilled whenever the response message is received (i.e., the WaitResponse task is fulfilled) and the value of the result attribute corresponds to the one contained in the message.

Although the FT annotations are very expressive, in the typical cases only a limited amount of the expressive power of FT is actually used. For instance, pre-/post-conditions are typically propositional. However, in some cases it is useful to have the possibility of expressing more complex conditions. for instance, in the case of the post-condition of the ProvideInformation task we use temporal operator **G** to specify that this task can only be considered fulfilled once all information requests have been processed. This high-



Goal Dependency ReceiveAssistance **Mode maintain**
Depender Citizen **Dependee** HealthcareAgency
Fulfillment condition ∀ dr: DoRequest (
(dr.**actor** = **depender** ∧ **Fulfilled** (dr) ∧ dr.result) →
F ∃ rs: ReceiveService (rs.**actor**=**depender** ∧ **Fulfilled** (rs)))

Task DoRequest **Mode achieve**
Super BeingAssisted **Actor** Citizen
Attribute result:**boolean**
Fulfillment definition
∃ wa:WaitAnswer(wa.**super** = **self** ∧
Fulfilled (wa) ∧ (result ↔ wa.result))

Task InitialRequest **Mode achieve**
Super DoRequest **Actor** Citizen

Task ProvideInformation **Mode achieve**
Super DoRequest **Actor** Citizen
Fulfillment definition
G (∀ ir: InfoRequest(**Received** (ir) → ∃ i: Info(**Sent** (i)))

Task WaitAnswer **Mode achieve**
Super DoRequest **Actor** Citizen
Attribute result:**boolean**
Fulfillment definition
∃ r:Response(**Received** (r) ∧ (result ↔ r.result))

Figure 3. Formal Tropos specification.

level condition is appropriate for the requirements model even if a specific mechanism will need to be used in the implementation to actually check when information request messages are ended (see Section 4). We remark that some temporal constraints are implicit in the semantics of FT and do not need to appear explicitly as annotations. For instance, an implicit creation constraint for each sub-goal is that the parent goal has not yet been fulfilled — if the goal has been fulfilled there is no reason to create the sub-goal. Also the order in which the sub-tasks of a given composed task are invoked is implicit in the FT semantics.

4. Integrating Business Requirements and Business Processes

The Web service business process specification may be easily derived from the business requirements model. The FT model already contains several pieces of information that can be exploited to generate a BPEL4WS specification. For instance, it is possible to automatically generate the definition of messages, ports, and services for the business domains — these elements define the WSDL document associated to the BPEL4WS specification. The description of the process model has to be completed by defining the body of the business process corresponding to the task. In our framework, this is achieved by associating to the task a business process defined in the BPEL4WS language. For instance, the business process corresponding to the task of submitting a request is described by the BPEL4WS specification in Figure 4.

Besides the result variable, which is already present in the formal requirements specification, the process contains additional variables waitResponse, vRequest, vInfoRequest, vInfo, and vResponse. The process behaves as follows. First, an initialization step is performed, during which the variable waitResponse is set to true, and the message Request is prepared by setting its need field. The Request message is sent in the following ⟨invoke⟩ command.

In order to fulfill the requirement that all incoming information requests should be satisfied until an answer has been received by the health-care agency, a ⟨while⟩ loop is entered. Its body is repeated until variable waitResponse becomes false. The body of the loop consists of a ⟨pick⟩ instruction which suspends the execution of the process until a InfoRequest or a Response message is received. Every incoming information request, arrived with a InfoRequest message, is answered with a corresponding Info message. The emitted Info message refers to the query contained in the received InfoRequest message. If a Response message is received, then the result variable of the process is set to reflect the result field of the received message. When this message is received, the citizen does not expect any other

```
<sequence name="DoRequestBody">
  <assign name="Initialization"
    event="Create ir:InitialRequest(ir.super=self)">
    <copy>
      <from expression="true()"/>
      <to variable="waitResponse"/>
    </copy>
  </assign>
  <invoke operation="oRequest" inputVariable="vRequest"/>

  <empty name="PhaseSwitch"
    event="Fulfill ir:InitialRequest(ir.super=self) &
           Create pi:ProvideInformation(pi.super=self)"/>

  <while condition="getVariableData('waitResponse')">
    <pick name="WaitMessage">

      <onMessage operation="oInfoRequest"
                 variable="vInfoRequest">
        <reply operation="oInfo" variable="vInfo"/>
      </onMessage>

      <onMessage operation="oResponse" variable="vResponse"
        event="Fulfill pi:ProvideInformation(pi.super=self) &
               Create wa:WaitAnswer(wa.super=self)">
        <assign name="LeaveLoop">
          <copy>
            <from expression="false()"/>
            <to variable="waitResponse"/>
          </copy>
          <copy>
            <from variable="vResponse" part="result"/>
            <to variable="result"/>
          </copy>
        </assign>
      </onMessage>

    </pick>
  </while>

  <empty name="DoRequestFulfilled"
    event="Fulfill wa:WaitAnswer(wa.super=self)"
    constraint="Forall wa:WaitAnswer(wa.super=self →
                G(wa.result↔self.result))"/>
</sequence>
```

Task InitialRequest
Mode achieve
Super DoRequest
Actor Citizen

Task ProvideInformation
Mode achieve
Super DoRequest
Actor Citizen
Fulfillment definition
G (∀ ir: InfoRequest(**Received** (ir) →
∃ i: Info(**Sent** (i)))

Task WaitAnswer
Mode achieve
Super DoRequest
Actor Citizen
Attribute result:**boolean**
Fulfillment definition
∃ r:Response(**Received** (r) ∧ (result ↔ r.result))

Task DoRequest
Mode achieve
Super BeingAssisted
Actor Citizen
Attribute result:**boolean**
Fulfillment definition
∃ wa:WaitAnswer(wa.**super** = **self** &
Fulfilled (wa) ∧ (result ↔ wa.result))

Figure 4. BPEL4WS process for task DoRequest of actor Citizen.

information requests and the PorvideInformation task can be considered completed. Therefore, variable waitResponse is set to false, so that the ⟨while⟩ loop is left.

Some additional attributes, which are specific for FT, are added to the BPEL4WS commands. These attributes are used to connect the evolution of the BPEL4WS process with the evolution of the requirements model. The event attributes describe which sub-tasks of DoRequest are supposed to be created or fulfilled in the requirements model when a given point is reached in the BPEL4WS code. For instance, sub-task InitialRequest is created during the initialization step and is fulfilled after the Request message has been sent (the BPEL4WS command ⟨empty⟩ is used to place this fulfillment event in the right position of the process). The constraint attributes define additional constraints between the requirements layer and the process layer. They are typically used to define the values of the attributes of the sub-tasks. For instance, the constraint attribute of Figure 4 binds the value of attribute result of the WaitAnswer sub-task to the value of variable result of the BPEL4WS process.

5. Formal Verification

The framework here proposed enables for several formal verification activities over the business requirements and business processes models. First, it is possible to verify the requirements model, e.g. by checking its consistency, or by verifying it against properties describing behavior that the model is supposed (or not supposed) to exhibit. This permits to discover inconsistencies and errors in the earliest phases, before having an actual implementation. As far as business processes are concerned, further kinds of analysis can be thought of. For instance, we can check for absence of deadlocks or livelocks in the devised protocol among the different parties involved in the business process. That is we can verify that the described process never blocks or it is never stuck in a loop. We can also verify business processes against business requirements and strategic goal models, thus providing an evidence that the given process actually implements and fulfills the requirements. Finally, we can verify whether the BPEL4WS protocol is consistent

with the published one. Moreover, in the inter-enterprise applications, not only separate business processes should be analyzed but also process compositions, for instance represented with BPEL4WS code. The implementation of these verification activities is still ongoing work.

In this section we present the T-Tool, a verification tool able to deal with FT specification, and we show how the functionalities it provides can be used to tackle some of the verification problems we envisaged.

5.1. The T-Tool Verification Tool

The T-Tool [9] supports the kinds of formal analysis described previously. Here we highlight some of its functionalities and we refer the reader to [9] for additional details. The T-Tool is based on finite-state model checking [5]. Model checking allows for an automatic verification of a specification with the generation of (counter-)example traces to witness the validity (or invalidity) of the specification. A limit of finite state model checking is that it requires a model with a finite number of states. Thus an upper bound on the number of class instances has to be specified in order to perform model checking.

The T-Tool input is an FT specification along with parameters that specify the upper bounds for the FT class instances. On the basis of this input, the T-Tool builds a finite model that represents all possible behaviors that satisfy the constraints of the specification. The T-Tool then verifies whether this model exhibits the desired behaviors. The T-Tool allows for different verification functionalities including interactive animation of the specification, automated consistency checks, and validation of the specification against possibility and assertion properties. Through animation, the user can generate valid scenarios for the specification by interacting with the T-Tool and incrementally extending a partial evolution of the model. Animation allows for a better understanding of the specified business domain, as well as for the identification of trivial bugs and missing requirements that are often taken for granted. Consistency checks are standard checks to guarantee that the FT specification is not self-contradictory. Inconsistent specifications occur quite often due to complex interactions among constraints in the specification, and they are very difficult to detect without the support of automated analysis tools. Consistency checks are performed automatically by the T-Tool and are independent of the application domain. Assertions properties describe conditions that should hold for all valid evolutions of the specification, while possibility properties describe conditions that should hold for at least one valid evolution. The verification phase usually generates feedback on errors in the FT specification and hints on how to fix them.

The T-Tool performs the verification of an FT specification in two steps (see Figure 5). In the first step, the FT specification is translated into an Intermediate Language (IL) specification. In the second step, the IL specification is given as input to the verification engine that is responsible of the actual verification. The T-Tool provides the user with two different verification engines built on top of state-of-the-art model checkers, that is the NuSMV [4] symbolic model checker and the SPIN [10] explicit state model checker.

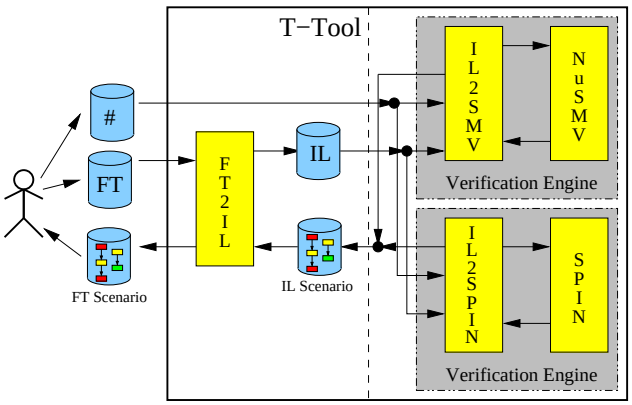


Figure 5. The T-Tool framework.

5.2. Verification of Business Requirements

Business requirements model enables for different kinds of verification to be carried out. In particular, the business requirements model can be automatically verified for consistency. Among the different kinds of verification the T-Tool provides are noticeable the verification of the possibility to create different goals, and to fulfill them thus showing that the specification is not over-specified and different goals do not conflict with each other.

Besides this, our framework allows business analyst to specify queries on the model in the form of properties that the requirements model is supposed to satisfy. We distinguish between **Assertion** properties, which describe conditions that should hold for all valid evolutions of the specification, and **Possibility** properties, which describe conditions that should hold for at least one valid evolution. Figure 6 reports an excerpt of desired properties for the considered case-study. Possibility P1 aims at guaranteeing that the set of constraints of the formal business requirements specification allows for the fulfillment of the task of doing a request in some scenario of the model. Assertion A1 requires that it is not possible for the citizen to fulfill its goal of receiving assistance services unless a positive response to a request from the health-care agency has been received. Finally, assertion A2 requires that the task of doing a request is eventually fulfilled under the condition that there is

Possibility P1
∃ dr: DoRequest (**Fulfilled** (dr))

Assertion A1
∀ c: Citizen (
∃ r: Response (**Received** (r) ∧ r.receiver = c ⟶ ¬ r.result) ⟶
∃ rs: ReceiveService (rs.**actor** = c ⟶ ¬ **Fulfilled** (rs)))

Assertion A2
∀ dr: DoRequest (
∃ ra: ReceiveAssistance (ra.**depender** = dr.**actor** ∧ **Fulfilled** (ra)
∧ ∀ r: Request (r.**sender** = dr.**actor** ⟶ r.**receiver** = ra.**dependee**))
⟶ **F Fulfilled** (dr))

Figure 6. Formal Tropos properties.

a health-care agency that is bounded to provide assistance to the user and, the citizen sends the request to that particular health-care agency.

All the properties in Figure 6 are satisfied on the final version of the business requirements model of the considered case-study. However, this result has required several revision steps, where both the model and the properties have been adjusted to capture the intended behaviors of the domain. For instance, assertion A2 had a crucial role in the process of precisely defining the mutual expectations incarnated by dependency ReceiveAssistance, and captured by the fulfillment constraints specified for this dependency as it can be seen in Figure 3.

It has to be remarked that, while verifying business requirements models, one cannot prove the business requirements specification is correct, since there is no reference model. However, the queries carried out on the business requirements specification can provide feedback on the captured behaviors and catch misunderstandings not trivial to be identified in an informal setting.

5.3. Verification of Business Processes

The definition of business processes, together with the bindings that link them to the corresponding tasks and messages in the formal requirements model, allow for different forms of verification. First, the given process can be checked for problems typical of a process specification, like the presence of deadlocks or livelocks. This is achieved by verification that the process specification will eventually complete on all its possible executions. For instance, in the case-study the citizen process can be blocked if it waits for the response from the health-care agency while the latter is not going to provide it for some reason (a deadlock). Or a health-care agency may request an additional information infinitely and the citizen process will stuck in this loop (a livelock). A further possibility consists of re-checking the formal queries defined for the requirements model (e.g., the properties in Figure 6) on the more detailed model. This is achieved by replacing a task of the FT specification (e.g task DoRequest) with the corresponding BPEL4WS process and by checking again the queries. Another possibility is checking that the refined model satisfies the requirements described by the **Creation** and **Fulfillment** constraints enforced in the requirements model for task DoRequest and its sub-tasks.

To support these kinds of verification, we have extended the T-Tool with a translation of BPEL4WS processes into the language of the verification engine chosen, i.e., into NuSMV or SPIN finite state machines. Transitions of these finite state machines are defined according to the semantics of the BPEL4WS constructs. Fairness conditions are added to the finite state machines to guarantee that the process eventually progresses whenever the next action to be executed is not blocked. In the case of the process in Figure 4, for instance, the only point where the process can be blocked forever is on the ⟨pick⟩ action, and only if no InfoRequest and Response messages are ever received. Finally, the event and constraint annotations are mapped into a set of temporal logic constraints that restrict the valid behaviors of the finite state machine.

At the time of writing we support only a restricted subset of BPEL4WS. Some restrictions rule out those operators that the verification techniques currently provided by the T-Tool are not able to deal with. In particular, the models to be verified have to be finite state, thus the usage of all those BPEL4WS constructs that may lead to an infinite number is restricted (e.g. interpretation of types of variables or creation of processes and compensation/fault handlers, which may lead to an infinite number of active instances). Another restriction concerns the time constructs of BPEL4WS (e.g., alarms and timeouts), that our verification tool is not able to manage. We are working to include other constructs (like flow, event-handlers or correlations) that are currently not supported, but that can be easily integrated.

The verification applied to the BPEL4WS process of Figure 4 pointed out some inconsistencies among the constraints of the requirements specification and the BPEL4WS process definition. For instance, the fulfillment definition of the DoRequest task (see Figure 4, right part) requires that the value of the ⟨result⟩ variable in this task should be equivalent to the value of the corresponding variable in the WaitAnswer task. In the BPEL4WS code (see Figure 4, left part) the value of this variable is copied directly from the Response message received. Thus, the corresponding variable of the WaitResponse task remains unchanged. This is shown in the counterexample represented in the Figure 7. The counterexample represents the message sequence chart of the execution of the model. The citizen creates the DoRequest task which generates the InitialRequest subtask where the request is sent to the health-care agency. In response, the latter creates tasks ReceiveRequest and EvaluateRequest. Then the subtask ProvideAnswer is created where the response message is created with positive result (result=1).

The DoRequest process receives the message and created the subprocess WaitAnswer. The value of its variable result is initially false. Then the value of the variable result in the DoRequest process is assigned to value contained in the received message. Thus, the values of the variables result of the DoRequest task and WaitAnswer do not coincide. The constraint become true if we copy the content of the message to the variable result of the WaitAnswer task.

As another example, if we modify the code of the process, e.g., by disallowing the reception of one of the two messages in the ⟨pick⟩ command, then the verification detects problems. If we disallow the reception of the InfoRequest message, the assertion A1 is violated. Indeed, if the health-care agency is requesting some information, the citizen is not able to answer to the request and the agency will not provide response to the citizen. Thus, it is impossible to fulfill the DoRequest. However, the ReceiveAssistance dependency still may be fulfilled, as it may be seen from the specification. In this case, the verification tool provides a counter-example similar to one represented in Figure 7. If we disallow the reception of the Response, even the possibility P1 becomes false. Indeed, if we do not receive the response, it is not possible to fulfill DoRequest.

The verification described so far deals mainly with the orchestration aspects of the Web service composition. However, other kinds of analysis may be applied in this framework. In particular, the choreography model of several participants may be verified against mutual expectations and requirements or even against global domain requirements on their composition.

We also remark that the approach described in this paper allows also for another kind of verification. Namely, in order to check that the process model is correct, one can show that it is equivalent to the requirements model according to a suitable behavioral equivalence. The current T-Tool verification engines, however, does not support yet this kind of verification.

6. Related Work

In current practice, business requirements and their refinement into business processes is done informally, or using semi-formal notations. For instance in [11] a method providing a model-driven transformations to design business processes is presented. The business views (requirements) are expressed in ADF [6] or in UML2 activity diagrams, which focus on the flow of information and control among the different activities, whereas the implementation is specified in BPEL4WS. In other frameworks process and workflow modeling languages like BPML [2] or XPDL [17] are used to model abstract and executable business processes and the supporting entities. All these formalisms are at the activity level according our requirements model (see Figure 2). This level of description of business requirements is very relevant but it is insufficient to capture the complexity of requirements of a distributed business process domain and to serve as a basis for automatic composition and verification of business processes. On the contrary, our approach allows to integrate the rationale behind the developed business process with its composite structure, thus allowing for capturing not only "how" the process is built, but also "why" the process is structured in a certain way.

A framework similar in spirit to the one proposed in this paper has been presented in [16], that addresses the problem of using UML2 for modeling business strategy and business processes. The framework focuses on the modeling problem without considering the validation of business requirements or the verification of business processes against business requirements. In our framework these kinds of analysis are aimed to support the modeling process and allow for identification and elimination of errors and conflicts in different development phases.

Several different formal specification notations are applied to the specification of the structure and the desirable behavior of software system. For instance, the Darwin [13] architecture description language describes the structure of the system in terms of components, interfaces and connectors. Moreover, it allows for the formal specification of the behavior of the system. The requirements-oriented and agent-oriented notations provided by Tropos seem more adequate for describing business requirements than the architecture-oriented notations provided by Darwin and similar languages.

As far as verification of business processes is concerned, the work described in [12] shows how to encode in the language of the NuSMV model checker business processes and how to proceed in refining the business processes to satisfy requirements properties. However, this pioneering work lacks of the link among the business requirements and the resulting business processes thus making it difficult to trace back the results obtained by the verification to business requirements. In [7, 8] a model-based approach for verifying Web service compositions is discussed. This approach provides for early-verification of properties created from design specifications and implementation models. Specifications of the design are modeled as UML Message Sequence Charts, which then are compiled into Finite State Process notation (FSP) for formal reasoning. BPEL4WS implementations are mechanically translated to FSP to allow for checking equivalence among the implementation and the UML specification. The approach is supported by a suite of cooperating tools for specification, formal modeling and trace animation of the composition of work-flows. Similarly to the other approaches, this work lack of the strategic analysis phase and of the link among requirements and the actual implementation.

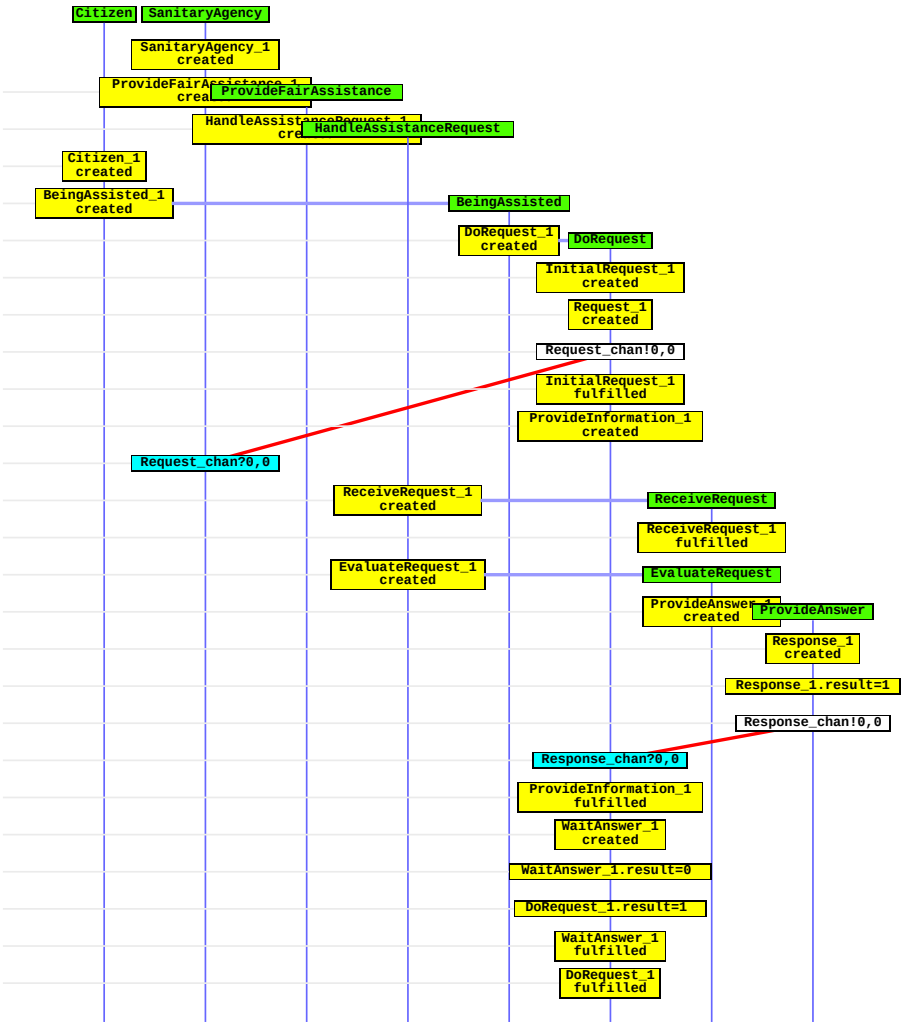


Figure 7. A counter-example generated by T-Tool.

Distinguishing feature of the approach presented here is that we start from a higher-level, strategic domain model, where processes as such are represented at a very abstract level and other types of requirements – for instance, general business rules on resource usage or engagement with other partners – can be easily represented. This gives us more flexibility in composing processes, and let us perform a wider range of verifications than directly starting from a business process or from elementary service definitions.

7. Conclusions and Future Work

In this paper we proposed a methodology based on an extension of Tropos for modeling business requirements, starting from strategic goals and constraints that are further refined and operationalized into business processes to achieve these goals and to satisfy the constraints. The methodology is also supported by a formal representation of the business requirements and business processes that enables for several kind of verification activities that constitute the basis for the automatic composition of distributed business processes. The verification activities are supported by a tool, the T-Tool, based on model checking, which allows to verify and suggest possible fix to the models as to satisfy the requirements.

The work presented here is our first step towards a long term vision where formal techniques are applied during the entire life cycle of services, from requirements analysis to execution. In the short term, we plan to extend the requirements specification language we described to bet-

ter capture the needs of the applicative domain, e.g., to provide for a better focus on activity level description of business requirements and to allow for a better integration business processes in the requirements model. Moreover, we plan to experiment with model checking tools different from the one provided by the T-Tool for being able to cope with all the verification tasks we envisaged. In the longer term, we will investigate how to improve the generation of BPEL4WS process skeletons in order to capture more details from the business requirements model, such as the type of long-term business transaction that is required. An improved BPEL4WS process should also enable an execution engine to relate faults and exceptions to specific goals or requirements of the domain model, in order to take appropriate action or provide feedback to the user.

On the longer term, we intend to integrate in our framework techniques supporting the change management. More precisely we intend to complement the verification techniques described in this paper, which are able to detect when changes in the requirements break the system, with automated code synthesis and code adaptation techniques, which are able to react to these changes. In particular, we will exploit synthesis techniques like the ones described in [15].

References

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *Business Process Execution Language for Web Services Version 1.1*, 2003.

[2] A. Arkin. *Business Process Modeling Language, Version 1.0*, November 2002.

[3] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the Tropos project. *Information Systems*, 27(6):365–389, 2002.

[4] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. of Computer Aided Verification Conference*. Springer, 2002.

[5] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[6] E. Deborin. *Continuous Business Process Management With Holosofx BPM Suite and IBM MQSeries Workflow*. IBM Redbooks, 2002.

[7] H. Foster, S. Uchitel, J. Magee, and J. Kramer. LTSA-BPEL4WS: Tool Support for Model-based Verification of Web Service Compositions. Technical report, Imperial College London, 2003.

[8] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service compositions. In *Proc. of the 18^th International Conference on Automated Software Engineering (ASE'03)*, 2003.

[9] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in Tropos. *Requirements Engineering*, 9(2):132–150, 2004.

[10] G. J. Holzmann. *The Spin Model Checker, Primer and Reference Manual*. Addison-Wesley, Reading, Massachusetts, 2003.

[11] J. Koehler, R. Hauser, S. Kapoor, F. Y. Wu, and S. Kumaran. A model-driven transformation method. In *Proc. of the Seventh International Enterprise Distributed Object Computing Conference (EDOC'03)*. IEEE Computer Society, 2003.

[12] J. Koehler, G. Tirenni, and S. Kumaran. From business process model to consistent implementation: A case for formal verification methods. In *6th International Enterprise Distributed Object Computing Conference (EDOC 2002)*. IEEE Computer Society, 2002.

[13] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *Proc. of the 5th European Software Engineering Conference, ESEC'95*. Springer-Verlag, 1995.

[14] C. Peltz. Web Services Orchestration and Choreography. *Web Services Journal*, 2003.

[15] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring web service composition. In *Proc. 11th Int. Conf. on Artificial Intelligence: Methodology, Systems, Architectures*, 2004. To appear.

[16] A. Vasconcelos, A. Caetano, J. Neves, P. Sinogas, R. Mendes, and J. M. Tribolet. A framework for modeling strategy, business processes and information systems. In *5th International Enterprise Distributed Object Computing Conference (EDOC 2001)*. IEEE Computer Society, 2001.

[17] Workflow Management Coalition. *Workflow Process Definition Interface – XML Process Definition Language, Version 1.0 Final Draft*, October 2002.