Modeling and Analyzing Context-Aware Composition of Services

Enzo Colombo[1], John Mylopoulos[2], and Paola Spoletini[1]

[1] Politecnico di Milano, Dipartimento di Elettronica e Informazione,
Via Ponzio 34/5, 20133, Milano, Italy
{colombo, spoleti}@elet.polimi.it
[2] Dept. of Computer Science, University of Toronto,
40 St. George Street, Toronto, Canada M5S 2H4
jm@cs.toronto.edu

Abstract. Service-oriented modeling and analysis is a promising approach to manage *context-aware* cooperation among organizations belonging to the same value chain. Following this approach, a value chain is modeled as a composition of services provided by different partners and coordinated in a way that their interactions can be reorganized according to changes in the environment. However, so far, most of the research work in this area has been focused on the design of architectures handling service discovery, compatibility and orchestration. Little attention has been given to the specification and verification of context-aware composition of services during the requirement engineering process. The goal of this paper is to fill this gap through a methodological approach based on the strict coupling between a social and a process model. The methodology is discussed through a simple example.

1 Introduction

Industrial districts consist of a number of enterprises, often small-to-medium (SME), that are physically close. These enterprises often collaborate through short-term projects to deliver products and services. In such a setting, enterprises strive to exploit flexible forms of collaboration with their business partners as a means to extend the boundaries of their planning activities, increase performance through cooperation and reduce TCO (Total Cost of Ownership). Industrial districts often include alliances, temporary or permanent, between two or more legal entities that exist for the purpose of furthering business or social objectives without causing the participants to lose their autonomy. In general, this cooperative environment is characterized by organizations that own heterogeneous information systems, with their own processes, procedures, data schemes, internal roles and responsibilities.

As a consequence, industrial districts represent an ideal environment for the implementation of a cooperative environment that supports the automation of inter-organizational business processes through the *logical composition* of distributed services representing public views on organizations' private workflows.

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 198–213, 2005.

In this context, ebXML is an example of a stable architectural solution that provides a specification language and an architecture shifting the logic of composition from information to service exchange [14].

However, ebXML does not support inter-organizational business processes that are context-aware in the sense that they are run-time customizable, i.e., they can readily adapt their structure according to feedbacks from the environment. For example, a previous agreement cannot be re-negotiated during the execution of a collaborative activity and a partner cannot be automatically replaced when cooperative goals are not fulfilled. In order to overcome these limitations, researchers and practitioners have focused much effort on implementing service-oriented architectures supporting context-aware collaborations among organizations [17,10]. In such settings, an inter-organizational process is implemented through a composition of services supplied over multiple channels by different actors. In particular, a composition of services describes the relationships among cooperating organizations according to a global, neutral perspective, in terms of valid *control* and *coordination* mechanisms. Moreover, a service composition is usually public, since it specifies the common rules defining a valid interaction among distributed business processes.

Unfortunately, the fruits of this research on context-aware applications does not have counterparts in methods, models and tools supporting the requirements engineering process. Indeed, according to [3], the conceptual modeling and analysis of context-aware composition of services is in its early stage even if this is the phase where the most and costliest errors are introduced to a design.

The goal of this work is to present a methodological framework that supports the conceptual modeling and formal analysis of requirements for context-aware service compositions through a social and a complementary process perspective. The paper also explores how modelers can analyze different process alternatives complying with the same social specification. Finally, our approach supports the formal verification of critical properties of a service composition (e.g., termination, structural soundness and achievement of shared goals). This work represents therefore a first step toward the design of service compositions aligned with different requirements policies.

In the remainder of the paper, we first motivate this work in terms of the state of the art. Then in Section 3, we define a set of different requirements policies adopted from the autonomic computing literature [9,13] that modelers can adopt during the requirements analysis and process specification. Section 4 discusses the requirements analysis process supporting the implementation of a composition of services. Finally, Section 5 discusses an example highlighting how our model formalizes service compositions with respect to different requirements.

2 Related Work

Most of the current work on context-aware composition of services is focused on service orchestration, discovery and semi-automatic management of compositions. For example, a theoretical model supporting service orchestration

through colored Petri nets is proposed in [12]. In particular, this work proposes a novel formal approach to the distribution of control responsibilities among different actors.

Moreover, formal models of service compositions supporting e-service discovery and composition are discussed in [1,18]. The work of Bultan et al. is mainly focused on providing a model of compositions for detailed design. Under this framework, individual services communicate through asynchronous messages and each service maintains a queue for incoming messages. Moreover, a global *watcher* keeps track of messages as they occur. However, this work pays little attention to the problem of specifying and analyzing service compositions, even though this is a key factor to improve collaboration among organizations. Notice that these modeling techniques are particularly important within industrial districts where the final output of a composition must comply with strategic goals shared among different organizations. Moreover, the violation of goals requires compensation actions aimed at leading the composition to a consistent state.

A promising starting point for a methodology supporting the specification of context-aware composition of services is the adoption of a social model. Indeed, this model facilitates goal refinement, the discovery of goal interactions, and the identification of services that can contribute to their achievement. Moreover, social models are consistent with coordination theory that constitutes the conceptual background for modeling service compositions [11]. Requirements specification through social models is discussed within the Tropos project, where the i^* model for early and late requirements analysis is discussed and formally defined. The i^* framework supports the modeling of social relationships among actors and has been widely experimented within the context of Multi-Agent System (MAS) development [3]. However, social specifications alone are inadequate for modeling control and coordination mechanisms. In particular, they lack a formal semantics to represent the standard and exceptional control flow for the actions constituting a service composition. Accordingly, i^* needs to be supplemented in order to be adopted in our particular application domain.

3 Policies for Context-Aware Service Composition

In the following, we define a core set of policies that modelers should evaluate during the requirements engineering process associated with the specification and verification of context-aware composition of services. This core set involves a level of self error detection, i.e. *controllability*, that defines the strategies to identify anomalous situations within a composition, and two levels of self-management, *flexibility* and *adaptability*. Flexibility (also, self-repair) concerns the management of problems repaired through the specification of ad-hoc compensation flows, while adaptability (also, self-configuration) addresses cooperation scenario changes when the same problem occurs over time. It should be noted that self-repair and self-composition are generally acknowledged as key features of autonomic systems [13].

Flexibility. Flexibility is referred to as the run-time management of service self-repair intended to bring a composition in a consistent state at the lowest cost and it is formalized according to three dimensions of analysis: *automation level*, *compensation classes* and *sparsity*.

Automation level is concerned with the degree of human intervention in conducting self-repair. We recognize three levels of intervention: automatic, manual and semi-automatic. If the system can self-repair by itself in the presence of anomalous events, the automation level is automatic, while if it only provides monitoring capabilities the automation level is manual. Finally, if a system does require some input to perform a compensation action, the automation level is semi-automatic.

Compensation actions are distinguished into five classes that, as discussed in [16], represent an exhaustive set of tasks that organizations may implement to return a composition to a consistent state.

- *Delay class* calls for simply waiting a predefined time interval hoping that the anomalous event is resolved; for example, missing information received after waiting beyond the due date.
- *Informative class* calls for actions that communicate a particular anomalous state of affair; for example a violation is notified to a business partner.
- *Re-negotiatiation class* involves either relaxation or tightening of goals and constraints a result of process failures.
- *Re-execution class* involves the re-execution of one or multiple services, possibly starting the execution of the whole process.
- *Re-transact class* involves the re-execution of the entire composition with other potential business partners. This kind of actions always involves the failure of the current composition and, possibly, the replacement of one or more process partners.

Sparsity formalizes where compensation actions take place with respect to where the violation of goals occurs [2]. When the compensation is executed by the business actor that detects the violation, the compensation is called centralized. On the other hand, when the action is executed elsewhere it is called delegated. A delegated compensation can be based on either a centralized or a delegated decision. When the actor raising the anomalous event specifies the compensation that its business partner should perform, the decision is centralized, otherwise it is delegated. Moreover, a delegated compensation can be deterministic or not depending on the knowledge of the identity of the business partners involved in the composition. A typical example of non-determinism is the delegation of a compensation action to any actor that plays a given role within the system. Finally, a compensation is participative if it is performed by more than one actor. For example, re-negotiation is intrinsically participative since it requires to establish a new agreement between two or more counterparts.

Controllability. During a service composition, anomalous events are detected and communicated by control activities whose aim is to evaluate the fulfillment of goals. Controllability concerns the level of visibility on the private business

process that implements a service, or the localization of control activities. Notice that, in our environment, control activities typically monitor quality of service goals (for example, service lead-time, productivity and use of resources).

Controllability is defined through two dimensions of analysis: service view and control policy. In service compositions such as the purchase of commodities by an occasional buyer, control is typically targeted to the end of the service with no intermediate checks during service execution. This view can be seen as black box since control is only possible when service outputs are delivered. Conversely, when control is possible on different activities during service execution, the service provides a public view on the private production process (i.e., grey-box).

Moreover, three control policies can be implemented when a service composition takes place. If control activities are performed where operating activities are executed, control is said to be centralized. On the other hand, control is delegated when control activities are performed elsewhere. Finally, if control activities are performed where operating activities are executed and repeated elsewhere, the control policy is redundant.

Let us consider a scenario that involves a service supplier and an occasional buyer. The former always monitors service lead-time since it have to guarantee an high quality of service (a violation of this commitment reduces the reputation of the buyer). The latter monitors the same attribute since it does not trust the supplier completely. This short-term relationship represent a simple case of redundancy since control is repeated by the buyer. However, we note that these two control activities could return different results if compared each other since service lead-time measured by the supplier could not consider network delays. As a consequence, redundant does not mean superfluous.

Adaptability is concerned with modifications of the standard and exceptional behavior of a composited process depending on the environment within which the composition is deployed. The environment is modeled through *(i)* the set of organizations involved within a composition (i.e., stakeholders) and through *(ii)* their goals over time.

In particular, adaptability is required when stakeholders' goals are repetitively violated over time. According to the stakeholder dimension, a designer may want to model different compositions as a function of the actors participating in the cooperative process. For example, when a business-to-consumer relationship is deployed, a provider could require payment before service delivery. On the contrary, for business-to-business interactions, payment could be required after delivery. We note that the specification of this adaptive behavior requires the formalization of two roles, i.e., corporate and retail. A stakeholder could be also modeled through either the channel or the device used during a service composition and, as a consequence, the behavior of a composition may vary accordingly. For example, a device could be a desktop, a laptop or a mobile phone. A channel could be a Virtual Private Network (VPN), Internet, a Wireless LAN or the GSM network. For each channel a designer may want to consider

the bandwidth and the level of security of the channel (e.g., low, medium, high). Therefore, a composition may vary depending on channel since organizations may decide that strategic information provided by a given service can be shared on a VPN (high-security, high-bandwidth) but not when the same service is required over the Internet. Moreover, a composition with an information service provided for a laptop (e.g., querying a warehouse to check the availability of a product) can be simpler if compared with a composition modeled for a desktop.

As discussed before, adaptation is especially desired when stakeholders' goals are repetitively violated over time. In this context, modelers should identify different alternatives to adapt the composition in order to reduce the violation of their goals. In our framework, we identify a main composition and a set of alternatives corresponding to other configurations when a goal/softgoal is repetitively violated. As a consequence, a composition shifts from an alternative to another depending on nature and number of violations. We note that violations can be either interleaved or not depending on the policy that we adopt for counting anomalous events. If the counter is reset every time a desired behavior is reached, the policy is not interleaved, otherwise it is.

4 Domain Requirements Analysis of Services Composition

Figure 1 shows the methodological steps through which modelers can perform the requirements modeling and analysis for a service composition. These steps comply with coordination theory that provides a theoretical foundation [11]. In particular, the methodology consists of a social analysis, a process analysis and a verification phase. In the following we present each step. We note that a social representation of a composition could generate different scenarios with different business rules and, as a consequence, different process models. These alternatives are evaluated studying the impact of different specification policies (see Sect. 3) on strategic goals. The evaluation process is performed adopting the labeling notation proposed in the NFR framework. Labels are defined as follows: satisfied (✓), weakly satisfied ($\mathcal{W}^+$), undecided ($\mathcal{U}$), weakly denied ($\mathcal{W}^-$), denied (✗), conflict (⌁) [5].

4.1 Social Analysis

The social specification of a service composition is organized in the following steps:

- *Step 1.1.* Identification of market players and dependencies; this step determines the organizations involved in the composition and their business relationships;
- *Step 1.2.* Refinement of business relationships, i.e., the actual pruning of intentional elements according to control and coordination policies.

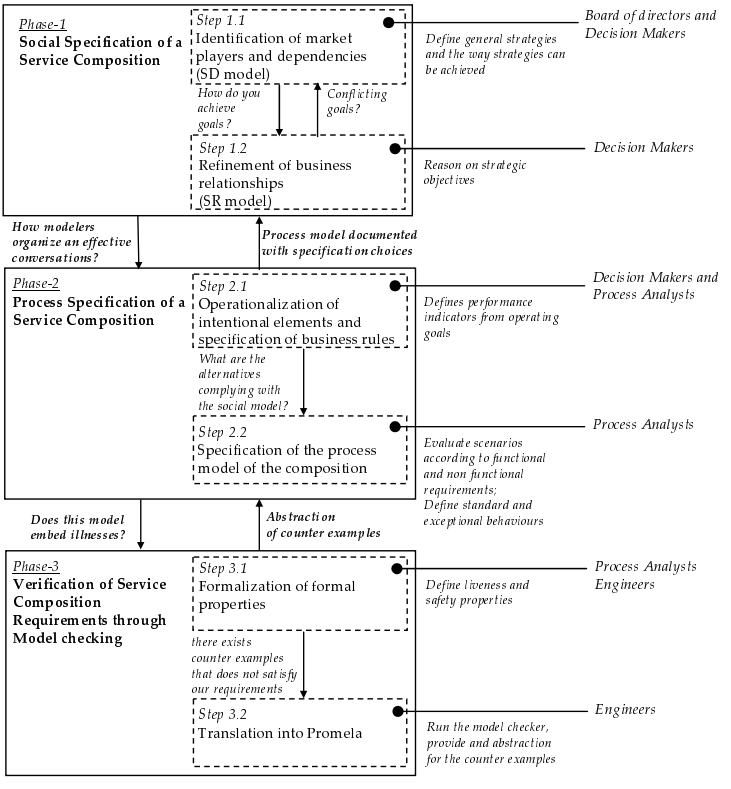

Fig. 1. Methodological steps supporting the analysis and specification of a composition of services

Our social analysis concerns a description of service composition that formalizes the strategy and the rationale of organizations interacting within a cooperative environment (i.e., *who*, *why* and *what*). In particular, directors and decision makers receive feasibility analysis and define the general goals that the composition should satisfy and the strategies through which these can be achieved. Then, general strategies are refined into more operating goals and the corresponding services fulfilling these goals are identified. The output of this step is an i^* social model of service composition and its level of detail is at the discretion of modelers.

In particular, our i^* specification embeds intentional elements such as softgoals, goals, service (i.e., a task in the traditional i^* notation) and information resources [3,6]. Goals represent requirements to be fulfilled (○= goal); softgoals are similar to goals but their fulfillment is not clearly defined (☁= softgoal). A service is a structured sequence of decisions and actions aimed at producing an added value transformation of inputs into outputs (⬭= service) and, finally, information resources represent inputs/outputs to services (□= resource).

Intentional elements are related to each other through Strategic Relationships (SR) and Strategic Dependencies (SD). The SD model concerns with the speci-

fication of social dependencies among organizations. In particular, an SD model is a graph where each node represents an organization and each link between two actors describes a dependency in terms of intentional entities. A dependency formalizes an agreement between two organizations, i.e. a depender and a dependee (depender – D – int. entity – D – dependee). The type of dependency defines the nature of the agreement. In particular, a goal (or softgoal) dependency represents the delegation of responsibility over the fulfillment of a goal (or softgoal) from a depender to a dependee. A service dependency represents the delegation of responsibility over the execution of a service from a depender to a dependee. With respect to goal (or softgoal), a service dependency is stronger since the depender also specifies how the service needed to fulfill a goal (or a softgoal) must be implemented. Finally, a resource dependency represents the need for an input that must be provided to a depender by a dependee. On the other hand, the SR model supports the refinement process of stakeholder goals through decomposition (—|–), contribution (→) and means-end (–▷) links. Directors and decision makers (see Figure 1) define their high-level goals and strategies and then, following a refinement process, elicit the set of services (and the corresponding resources) that should be performed to achieve their goals (and softgoals).

4.2 Process Analysis

The process specification of a service composition is organized according to the following steps:

- *Step 2.1.* Operationalization of intentional elements and specification of business rules managing either goals fulfillment or violation.
- *Step 2.2.* Specification of the process model of composition complying with both *(i)* the social model and *(ii)* the core set of policies that could be adopted when modeling context-aware compositions (see Sect. 3).

Our process analysis describes the control and coordination mechanisms of a service composition. In particular, decision makers receive a social model from the previous step and, together with process analysts, define the business rules modeling the standard and exceptional behavior of a service composition. Our approach to the transformation of a social model into business rules has been discussed in [6]. Moreover, business rules are specified according to ECA (event, condition, action) rules complying with the following semantics [4]:

Events are only of two types $End(sv), Begin(sv)$ where sv is a service, with the natural meaning of beginning and end of the service passed as argument.

Let S be a set of symbols representing actors, RO a set of symbols representing roles played by actors in S, G a set of symbols representing strategic goals, R a set of symbols representing information resources, X_t a set of discrete clocks and CH and DV sets of symbols representing respectively channels and devices used to supply a service in the conversation. A *condition* is a predicate p, that can be categorized in the following classes:

1. If p has the form $Achieved(g)$, $g \in G$, it is a *goal condition*.
2. If p has one of the forms $Fulfilled(a)$ or $Done(a)$, $a \in A$, p is called *compensation condition*.
3. If p has one of the forms $Actor(s)$, $Role(s, ro)$, $Device(dv)$, $Channel(ch)$, p is called *user condition*. $Actor(s)$ is satisfied when the current actor is $s \in S$; $Role(s, ro)$ is satisfied when the actor $s \in S$ plays the role expressed by $ro \in RO$; $Device(dv)$ is satisfied when the current device is $dv \in DV$ and $Channel(ch)$ is satisfied when the current channel is $ch \in CH$.
4. If p is a conjunction of predicates of the form $[\rho \bullet c]_t$, where $\bullet \in \{\leq, \geq, =, <, >\}$, $\rho \in X_t$ is a discrete clock, $c \in \mathbb{N}$ is a constant and the subscript t indicates a time measurement unit, it is a *temporal condition*.
5. If p has the form *(i)* $[\rho \bullet c]_t$, where $\bullet \in \{\leq, \geq, =, <, >\}$, ρ is a variable, c is a constant and the square brackets with the index t denote that ρ and c are of the same measurement unit t or *(ii)* $Received(x, s, r) \wedge x \in X$, where $r \in R$, $s \in S$ and X is a set of temporal conditions, p is a *resource condition*.

Actions can be composed by means of logical (i.e., $\neg$, $\vee$, $\wedge$) and *Sequence* operators. When actions are composed with $\vee$, the action to be enacted is selected non-deterministically. The *Sequence* operator involves the execution of a finite number of compensation actions in a sequence. However, compensation stops at the first successful compensation action in the sequence. Moreover, compensation actions are grouped into classes (see Sect. 3), i.e. *delay* (e.g., wait for, delay, ...), *informative* (e.g., notify, urge,...), *re-execute* (e.g., re-execute, skip,...), *re-negotiate* (e.g., relax, tighten,...) and *re-transact* (e.g. delegate execution,...).

For example, $Wait\_for([t_0, t_1], r)$ requires to wait for a resource within t_0 and t_1 time units, $Re\_execute([t_0, t_1], sv)$ requires the re-execution of a service sv, $Urge([t_0, t_1], sv, r)$ urges to the service sv the delivery of a resource r and $Relax$ $([t_0, t_1], sv, [\rho \bullet c]_m)$ requires to service sv the relaxation of the constraint $[\rho \bullet c]_m$.

Finally, business rules are then mapped into a process model, i.e. a particular instance of statechart [7] where transitions are labeled by the set of business rules defined so far and where states labels are defined as follows [4].

A state label l_q is a 5-uple $l_q =< sv, \{s_1, \ldots, s_n\} / \{ro_1, \ldots, ro_n\}, x, ch, dv >$, with $sv \in SV$, $s_i \in S\ \forall i \in [1 \ldots n]$, $ro_j \in RO\ \forall j \in [1 \ldots n]$, $x \in X$, $ch \in CH$, $dv \in DV$. The initial state q_0 has no label. Final state labels are modeled as $<$ [*commit*, *abort*, *pending*], null, null, null, null$>$.

We note that the symbol ξ in the action part of an ECA rule means that no action is performed during the transition from a state to another.

4.3 Verification Phase

The verification phase is organized as follows:

- *Step 3.1.* Formalization of safety and liveness properties [15] related to our process model.
- *Step 3.2.* Translation of the process model into the Promela language of the SPIN model checker [8].

This final step verifies that the process model is correct, or otherwise provides a counter example that points to specification inconsistencies.

Properties are generally defined by process analysts on the basis of requirements and then specified as LTL logic formulas by engineers. Hence, the process description of a composition of services C is accepted *iff* it satisfies a set of LTL formulas. Formally, let φ the conjunction of all LTL formulas, the process model is accepted *iff* $C| = \varphi$.

Our properties can be classified as follows:

- **Structural properties**, modeling the functional characteristics of a composition of services. Critical structural properties include the verification that a composite system is deadlock-free (i.e., absence of invalid end-states), that it does not embed infinite cycles and that each service belonging to the process is invoked (i.e., total functional coverage). More in general, structural properties model each functional expectation from a run of a composition of services involving a particular sequence of invocations, the ownership of each service and the device/channel used to deliver a service.
- **Temporal properties**, modeling time constraints of a composition. In particular, temporal requirements state that a service belonging to the composition can not be invoked in a time less then or equal to t. Moreover, we can also require that a service is not invoked before t.
- **Quality of Service (QoS) properties**, modeling the quality requirements of a composition. Critical QoS properties formalize strategic business goals whose fulfillment depends from the satisfaction of Service Level Agreement (SLA) parameters such as productivity, yield, price and throughput.

Accordingly, LTL formulas can formalize the following critical scenarios:

- a scenario involving a single property. For instance, we may require that the process lead-time is always constrained below a given threshold (i.e., $\Box(lead - time < threshold)$;
- a scenario involving dependencies among properties belonging to the same class. For instance, we may require that when a particular quality requirement is not fulfilled, the overall price of the composition must be below a predefined threshold (i.e., $\Box[(throughput \leq .5) -> \diamond(price < InitialPrice)]$);
- a scenario involving dependencies among properties belonging to different classes. In particular, assertions relating either temporal and QoS properties with structural properties are useful to validate scenarios involving the behavior expected from a composite system as a consequence of exceptions. For instance, we may require that the violation of a quality requirement always lead to a negotiation of the initial agreement and viceversa (i.e., $\Box[(throughput \leq .5) \Leftrightarrow \diamond\mathbf{Done}(negotiation)]$).

Notice that verification is possible since we generate non-deterministically all the possible values of temporal and QoS variables. These values are obtained by discretizing the domain of each variable into a finite number of significant values. In this way, we keep finite the number of alternatives. For sake of the

simplicity, the reader can assume that each state of the model is mapped into a Promela process and that transitions among states are represented through the exchange of messages between Promela processes. Under such sattings, the non-deterministic generation of a temporal and QoS variable associated with a service is implemented within its corresponding Promela process. An in-depth discussion of the performances of our model checking technique has been provided in [4].

5 Example

This section illustrates how the social and process models proposed in Sect. 4 support the specification of a service composition according to different degrees of flexibility, controllability and adaptability. Hence, our goal in this section is threefold. First, we provide an intuitive use of our specification models through a simple example. Second, we discuss how a single social specification can be mapped into multiple alternative process models. Finally, we show how model checking supports the identification of inconsistent behaviors in the process specification thus guiding modelers in their work.

Let us suppose that, within an industrial district, a buyer company buys laptop components on the market and supplies assembled laptops to a selected network of retailers. Moreover, this buyer company decides to control components before assembly since it aims at minimizing laptop malfunctions (i.e., errors). In order to reduce total costs, the buyer also aims at minimizing the interaction with the supplier. On the other hand, potential sellers within the district does not provide any visibility on their production process. The production process is thus private. However, during component delivery, they provide component technical features required by the buyer to make quality control. We note that this example precisely defines our requirements for controllability (i.e., control delegation, black-box control). We note that control is considered delegated since it is not implemented by the seller locally.

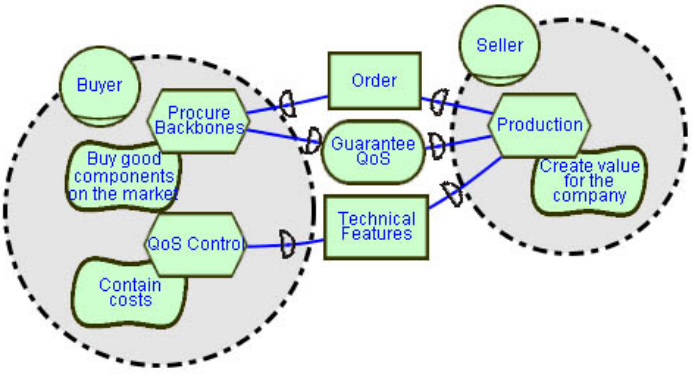

Fig. 2. Social model involving a buyer company and its laptop components supplier

Table 1. Contribution of different alternatives on strategic goals (NFR Analysis)

	Policy			Result	
	Flexibility	*Controllability*		*Subgoals*	*Goal*
1	×	black-box control ✓	delegated control ✓	minimize interactions $\mathcal{W}^+$ / minimize errors $\mathcal{W}^-$	Contain Cost $\mathcal{W}^+$
2	centralized decision ✓	black-box control ✓	delegated control ✓	minimize interactions × / minimize errors ✓	Contain Cost $\mathcal{W}^+$
3	delegated decision ✓	black-box control ✓	delegated control ✓	minimize interactions $\mathcal{U}$ / minimize errors ✓	Contain Cost ✓

However, there are several possible choices with respect to flexibility that need to be explored and compared during the design process. The social model associated with this cooperating scenario is shown in Figure 2.

At this stage, the buyer may want to evaluate the impact of different policies (see Sect. 3) on its high-level *Contain Costs* softgoal under the hypothesis that this goal is decomposed into *Minimize Errors* and *Minimize Interactions*. Table 1 studies the impact of controllability and flexibility on these softgoals through the NFR framework. In particular, flexibility impacts negatively on the *Minimize Interaction* softgoal but, on the contrary, it contributes positively to *Minimize Error*.

First of all, let us consider the simplest specification scenario, i.e. flexibility is not satisfied. This means that violations of the *Guarantee QoS* goal are not managed. The analysis of the NFR three shows that this configuration weakly satisfies the *Contain Cost* softgoal. In particular, the *Minimize Interaction* softgoal is weakly satisfied and the *Minimize Error* softgoal is weakly denied. If the buyer is happy with the adoption of a strategy resulting in a weak satisfaction of its high-level softgoal, the process model formalizing our cooperating scenario is shown in Figure 3(a). This specification presents a black-box delegated control where compensation is not implemented since when the QoS goal is violated the composition automatically aborts.

The second alternative (see Table 1) is intended to specify a cooperating scenario where the buyer wants to be sure that the compensation action raised by violations of the *Guarantee QoS* goal brings the composition in a consistent state. Accordingly, the buyer requires the implementation of a centralized decision, but this requirement denies the *Minimize Interactions* softgoal. On the other hand, the implementation of a centralized decision guarantees the satisfaction of *Minimize Errors*. The final result is weak satisfaction of *Contain Costs*, as with the first alternative. This means that the fulfillment of the *Minimize Errors* softgoal balances the structural complexity derived by more interactions.

Figure 3(b) shows the process model associated with our second alternative. With respect to the previous specification, from the perspective of controllability, our scenario is unchanged since control is black-box and delegated. However, the specification is a little bit more complex since compensation classes are introduced together with sparsity. Figure 3(b) enriches the scenario described in Figure 3(a) by allowing the re-execution of the production service when the QoS goal is violated. Moreover, since re-execution is allowed exactly once, if the QoS

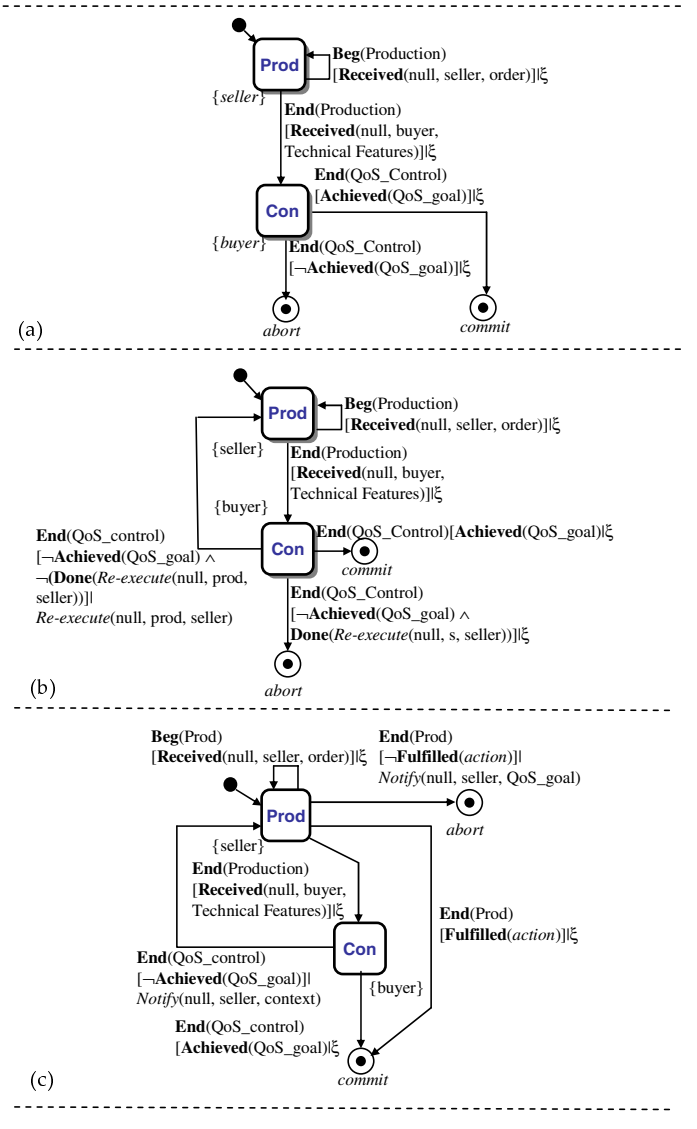

Fig. 3. Poor flexibility (a), centralized decision (b), delegated decision (c)

goal is still violated the composition aborts. Re-execution is performed according to the following guard condition: $\neg Achieved(QoS\_goal) \wedge \neg(Done(Re-execute(null, s, seller))$.

In summary, from the perspective of *flexibility*, this scenario is automatic, uses re-execute compensation class and compensation is delegated with a centralized decision. The compensation is delegated since it is executed by an actor different from the one raising the exception, i.e. the seller. Moreover, decision is centralized

since the compensation action is decided by the actor raising the exception (i.e., the buyer).

Finally, the third alternative in Table 1 studies the impact of flexibility on high-level softgoals in case of delegated decision. According to this scenario, the impact of flexibility on the *Minimize Interaction* softgoal improves from *break* to *hurt* [5]. As a consequence, the *Minimize Interaction* softgoal is not *denied* but *undecided*. Moreover, the *Minimize Errors* softgoal is still satisfied meaning that the *Contain Costs* softgoal is satisfied as well. Hence, with respect to previous two alternatives, this current alternative seems to capture a better compromise.

Figure 3(c) shows the process model associated with this third alternative. In this case, a notification is provided to the seller that will perform a corresponding compensation action. If the compensation fails the composition is aborted, otherwise committed. Since the service view is black-box, the buyer is not aware of the rules followed to compensate the violation. The buyer is only aware of the behavior of the composition, independently of whether the compensation fails or not.

Once the better policy has been identified, our last step is requirements verification. Indeed, checking that the behavior of the compensation is consistent with our requirements is a critical activity of our modeling process. Under this scenario, we have two main vital requirements for our composition of services.

- An instance of the composition always terminates: $\Box\Diamond(commit \vee abort)$.
- The composition commits either if the QoS goal is fulfilled or if its violation is successfully compensated:
  $\exists q_n \in F[l_n = \langle commit,\ null,\ null,\ null,\ null\rangle \rightarrow$
  $(\exists a : Action,\ qs : QoSGoal(Achieved(qs) \vee Fulfilled(a)))]$.

The analysis of the process model through model checking shows that *Prod* is an invalid end state [8]. In particular, the generated counter-example shows that the composition does not terminates into either commit or abort since our specification does not model what happens when the *order* information resource is not received. Hence, our model is enriched with a transition from *Prod* to *abort* labeled as following:
$Beg(Production)[\neg Received(null,\ seller,\ order)]|\xi$.

Moreover, a successive analysis shows the same problem for the *Con* state. However, in this case *Production* has been already executed and modelers want to avoid, if possible, an abort of the composition. As a consequence the process model is completed as follows:

- A self-loop on the *Prod* state is added to urge the provisioning of *Technical Resources* This transition is labeled as follows:
  $End(Production)[\neg Received(null,\ seller,\ TechnicalFeatures)]|$
  $Urge([1,3],\ buyer,\ TechnicalFeatures)$.
- A *pending* state modeling that control is given to a human operator is added to the composition to handle a failure of the *Urge* compensation action. The transition from *Prod* to pending is labeled as follows:
  $End(Production)[\neg Fulfilled(Urge([1,3],\ buyer,\ TechnicalFeatures)]|\xi$.

This new version of the original process model fully satisfies the two critical requirements formalized for our composition of services.

6 Conclusion and Future Work

This paper has presented a methodological framework that supports the modeling and formal analysis of service compositions extending the i^* social model adopted in Tropos [3] with a complementary process perspective. Moreover, this work has discussed a set of policies that designers should consider when shifting the attention from a social representation of the cooperative environment to one of the possible process scenarios. In summary, our proposal represents the first step toward the implementation of autonomic inter-organizational business processes, i.e., business processes that can self-repair, self-configure and self-tune on the basis of feedbacks from the environment [13]. Specifically, we envision an environment where several service compositions exist, but one is selected for execution. If there are problems with this execution, the system can self-repair or self-reconfigure by shifting to an alternative composition to improve its performance with respect to the fulfillment of stakeholder goals. Mechanisms for changing the composition on the basis of different types of feedback have not been studied yet in our work.

Future research direction will include the development of a theory of robustness for service compositions. In particular, we will study techniques to ensure that a composition behaves in a reasonable way even when part of the goals are inconsistent, implausible or unrealizable with the resources available.

References

1. T. Bultan, X. Fu, R. Hull, and J. Su. Composition specification: a new approach to design and analysis of e-service composition. In *Proceeding of the International Conference on the World Wide Web (WWW'03)*, ACM press, pages 403–410, 2003.
2. F. Casati and G. Pozzi. Modeling exceptional behaviours in commercial workflow management systems. In *Proceeding of the CoopIS/DOA/ODBASE 1999*, LNCS, pages 127–138, 1999.
3. J. Castro, M. Kolp, and J. Mylopoulos. Towards requirement-driven information systems engineering: the tropos project. *Information Systems*, 27:365–389, 2002.
4. A. Cherubini, E. Colombo, C. Francalanci, and P. Spoletini. A formal approach supporting the specification and verification of business conversation requirements. In *Proceeding of the IADIS International Conference on Applied Computing*, 2005.
5. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering Series*, volume 5. Kluwer Internationale Series in Software Engineering, 2000.
6. E. Colombo, C. Francalanci, and B. Pernici. Modeling cooperation in virtual districts: a methodology for e-service design. *International Journal of Cooperating Information Systems, Special Issue on Service Oriented Modeling*, 13(4):337–369, 2004.
7. D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Trans. on Soft. Eng. and Method.*, 5(4):293–333, 1996.

8. G.J. Holtzmann. *The SPIN Model Checker*. Addison-Wesley, 2004.
9. J. Kephard, M. Parashar, V. Sunderam, and R. Das, editors. *Proceedings of the International Conference on Autonomic Computing*, 2004.
10. Mais project. http://black.elet.polimi.it/mais/index.php.
11. T.W. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Comp. Surveys*, 26(1):87–119, 1994.
12. M. Mecella, F. Parisi-Presicce, and B. Pernici. Modeling e -service orchestration through petri nets. In *Proceedings of the 3nd VLDB International Workshop on Technologies for e-Services*, pages 38–47, 2002.
13. R. Murch. *Autonomic Computing*. Prentice Hall, 2004.
14. ebxml project. www.ebXML.org.
15. A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
16. W. R. Scott. *Organizations: Rational, Natural and Open Systems*. Prentice Hall, 1992.
17. Vispo project. www.casaccia.enea.it/vispo.
18. A. Wombacher and B. Mahlenko. Finding trading partners to establish ad-hoc business processes. In *Proceeding of the CoopIS/DOA/ODBASE 2002*, LNCS, pages 339–355, 2002.