Adaptation Goals for Adaptive Service-oriented Architectures*

Luciano Baresi and Liliana Pasquale

Abstract Service-oriented architecture supports the definition and execution of complex business processes in a flexible and loosely-coupled way. A service-based application assembles the functionality provided by disparate, remote services in a seamless way. Since the architectural style prescribes that all features be provided remotely, these applications adapt to changes and new business needs by selecting new partner services to interact with. Despite the success of the architectural style, a clear link between the actual applications —also referred to as service compositions— and the requirements they are supposed to meet is still missing. The embedded dynamism also imposes that requirements properly state how an application can evolve and adapt at runtime. The solution proposed in this chapter aims to solve these problems by extending classical goal models to provide an innovative means to represent both conventional (functional and non-functional) requirements and adaptation policies. To increase the support to dynamism, the proposal distinguishes between crisp goals, of which satisfiability is boolean, and fuzzy goals, which can be satisfied at different degrees; adaptation goals are used to render adaptation policies. The information provided in the goal model is then used to automatically devise the application's architecture (i.e., the composition) and its adaptation capabilities. The goal model becomes a live, runtime entity whose evolution helps govern the actual adaptation of the application. All key elements are exemplified through a service-based news provider.

Luciano Baresi and Liliana Pasquale
Politecnico di Milano, Dipartimento di Elettronica e Informazione, piazza L. da Vinci, 32 - 20133 Milano Italy, e-mail: `{baresi|pasquale}@elet.polimi.it`

* This research has been funded by the European Commission, Programmes: IDEAS-ERC, Project 227977 SMScom, and FP7/2007- 2013, Projects 215483 S-Cube (Network of Excellence).

1 Introduction

In these years, Service-oriented Architecture (SoA) has proven its ability to support modern, dynamic business processes. The architectural paradigm fosters the provision of complex functionality by assembling disparate services, whose ownership —and evolution— is often distributed. The composition, oftentimes rendered in BPEL [18], does not provide a single integrated entity, but it only interacts with services that are deployed on remote servers. This way of working fosters reusability by gluing existing services, but it also allows one to handle new business needs by adding, removing, or substituting the partner services to obtain (completely) different solutions.

So far, the research in this direction has been focused on proposing more and more dynamic service compositions, neglecting the actual motivations behind them. *How* to implement a service-based application has been much more important than understanding *what* the solution has to provide and maybe how it is supposed to evolve and adapt. A clear link between the actual architectures —also referred to as service compositions— and the requirements they are supposed to meet is still missing. This lack obfuscates the understanding of the actual technological infrastructure that must be deployed to allow the application to provide its functionality in a robust and reliable way, but it also hampers the maintenance of these applications and the alignment of their functionality with the actual business needs.

These considerations motivated the work presented in this chapter. We firmly believe that service-based applications must be conceived from clearly stated requirements, which in turn must be unambiguously linked to the services that implement them. Adaptation must be conceived as a requirement in itself and must be properly supported through the whole lifecycle of the system. It must cope with both the intrinsic unreliability of services and the changes imposed by new business perspectives. To this aim, we extend a classical goal model to provide an innovative means to represent both conventional (functional and non-functional) requirements and adaptation policies. The proposal distinguishes between *crisp* goals, the satisfiability of which is boolean, and *fuzzy* goals, which can be satisfied at different degrees; *adaptation* goals are used to render adaptation policies.

The information provided in the goal model is then used to automatically devise the application's architecture (i.e., the composition) and its adaptation capabilities. We assume the availability of a suitable infrastructure —based on a BPEL engine— to execute service compositions [5]. Goals are translated into a set of abstract processes (a-la BPEL [18]) able to achieve the objectives stated in the goal model; the designer is in charge of selecting the actual composition that best fits stated requirements. Adaptation is supported by performing supervision activities that comprise data collection, to gather execution data, analysis, to assess the application's behavior, and reaction —if needed— to keep the application on track. This strict link between architecture and requirements and the need for continuous adaptation led us to consider the goal model a full-fledged runtime entity. Runtime data trigger the countermeasures embedded in adaptation goals, and thus activate changes in the goal model, and then in the applications themeselves.

The rest of the chapter is structured as follows. Section 2 presents the goal model to render the requirements of these systems. Section 3 describes how goals are translated into service-based applications, along with the rules that guide the supervision at runtime. Section 4 illustrates some preliminary evaluation, Section 5 surveys related works and Section 6 concludes the chapter.

2 Goal Model

This section introduces the goal model adopted to represent requirements. Conventional (functional and non-functional) requirements are expressed by adopting KAOS [14], a well-known goal model, and RELAX [27], a relatively new notation for expressing the requirements of adaptive systems. The goal model is also augmented with a new kind of goals, adaptation goals, which specify how the model can adapt to changes. The whole proposal is illustrated through the definition of a fictious news provider called Z.com [8]. The provider wants to offer graphical news to its customers with a reasonable response time, but it also wants to keep the cost of the server pool aligned with its operating budget. Furthermore, in case of spikes in requests it cannot serve adequately, the provider commutes to textual content to supply its customers with basic information with acceptable delay.

2.1 KAOS

The main features provided by KAOS are goal refinement and formalization. Goal refinement allows one to decompose a goal into several conjoined subgoals (AND-refinement) or into alternative subgoals (OR-refinement). The satisfaction of the parent goal depends on the achievement of all (for AND-refinement) or at least one (for OR-refinement) of its underlying sub-goals. Goal decomposition can also be accomplished through formal rules [14]. The refinement of a goal terminates when it can be "operationalized", that is, it can be decomposed into a set of operations.

Figure 1 shows the KAOS goal model of the news provider. The general objective is to provide news to its customers (G1), which is AND-refined into the following sub-goals: Find news (G1.1), Show news to requestors (G1.2), Provide high quality service (G1.3), and Maintain low provisioning costs (G1.4) (in terms of the number of servers used to provide the service). News can be provided both in textual and graphical mode (see OR-refinement of goal G1.1 into G1.1.1 and G1.1.2). Textual mode consumes less bandwidth and performs better in case of many requests. Customer satisfaction is increased by providing news in a nice format and within short response times (see AND-refinement of goal G1.3 into G1.3.1 and G1.3.2). G1.3.1 is a soft goal since there is not a clear-cut criterion to assess it, that is, whether news are provided in a nice way.

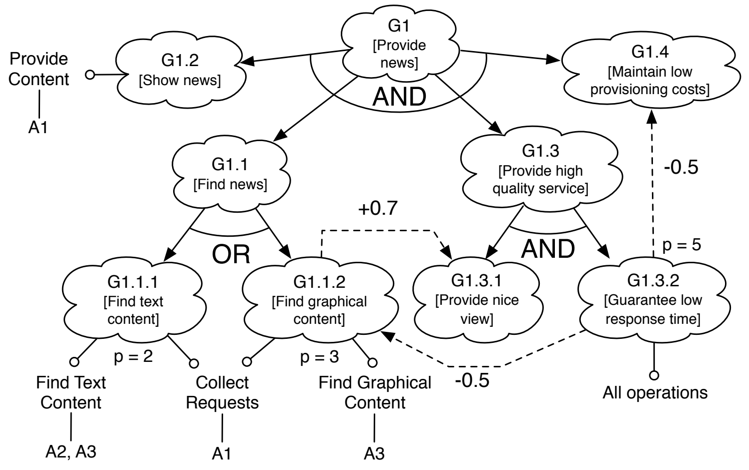

Fig. 1 The KAOS goal model of the news provider.

G1.1.1	$r : NewsReq,\ nc : NewsCollection,\ ReceiveRequest(r) \wedge$ $nc.keyword\ = \text{“”} \wedge nc.date = null \wedge (r.keyword \neq \text{“”} \vee r.date \neq null) \implies$ $\Diamond_{t<x}((\exists\, n \in nc.news :\ nc.keyword = r.keyword \vee nc.date = r.date) \wedge n.text \neq null)$
G1.1.2	$r : NewsReq,\ nc : NewsCollection,\ ReceiveRequest(r) \wedge$ $nc.keyword\ = \text{“”} \wedge nc.date = null \wedge (r.keyword \neq \text{“”} \vee r.date \neq null) \implies$ $\Diamond_{t<x}((\exists\, n \in nc.news : nc.keyword = r.keyword \vee nc.date = r.date) \wedge$ $n.text \neq null \wedge (\exists\, i \in n.images))$
G1.2	$nc : NewsCollection,\ n : News\ (\exists\, n \in nc.news) \implies \Diamond_{t<y}\ ShowNews(nc)$
G1.3.2	$r : NewsReq,\ nc : NewsCollection,\ ReceiveRequest(r) \wedge$ $nc.keyword\ = \text{“”} \wedge nc.date = null \wedge (r.keyword \neq \text{“”} \vee r.date \neq null) \implies$ $\Diamond_{t<RT_{MAX}}\ (ShowNews(nc))$
G1.4.1	$servers : int,\ servers \leq N_{MAX}$

Table 1 Definition of the example's goals.

Goals are associated with a priority depending on their criticality. For example goal G1.1.1 has lower priority ($p = 2$) than goal G1.1.2 ($p = 3$), since providing news in graphical mode is more important than providing news in text mode. Goals can contribute (either positively or negatively) to the satisfaction of other goals. This is represented in the goal model through contribution links —dashed lines in Figure 1— and an indication of the contribution ($x \in [-1, 1]$). For example, despite the graphical mode is slower, it positively contributes to the customer satisfaction (contribution link between goals G1.1.2 and G1.3.1). Short response times may require the adoption of the text mode to provide the news (see the negative link between goal G1.3.2 and goal G1.1.2) or may increase the provisioning costs since they may require a higher number of servers in the pool (see the link between goal G1.3.2 and G1.4).

Name:	**Collect Requests**
In/Out:	$r : ReceiveRequest,\ nc : NewsCollection$
DomPre:	$nc.state = default$
DomPost:	$nc.state = req\_initialized$
ReqPre:	$nc.keyword = \text{“”} \wedge nc.date = null \wedge (r.keyword \neq \text{“”} \vee r.date \neq null)$
TrigPre:	$ReceiveRequest(r)$
ReqPost:	$nc.keyword = r.keyword \wedge nc.date = r.keyword \wedge Collect(nc.keyword, nc.date)$
Name:	**Find Graphical Content**
In/Out:	$nc : NewsCollection$
DomPre:	$nc.state = req\_initialized$
DomPost:	$nc.state = news\_received$
ReqPre:	$nc.keyword = \text{“”} \wedge nc.date = null \wedge (r.keyword \neq \text{“”} \vee r.date \neq null)$
TrigPre:	$Collect(nc.keyword,\ nc.date)$
ReqPost:	$\forall n \in nc.news : n.keyword = nc.keyword \vee nc.date = n.date \wedge n.text \neq null \wedge (\exists\, i \in n.images : i.content \neq null) \wedge (\exists\, n \in nc.news)$
Name:	**Provide Content**
Input:	$nc : NewsCollection$
ReqPre:	$(\exists\, n \in nc.news)$
TrigPre:	$@(nc.state = news\_received)$
ReqPost:	$ShowNews(nc)$

Table 2 Definition of example's operations.

Goals are formalized in Linear Temporal Logic[2](LTL) [21] or First Order Logic (FOL). The definition of the leaf goals of Figure 1 is reported in Table 1. For example, goal G1.1.2 states that if the system receives a request for a given keyword and date, it must provide related news within x time units. Provided news must be about supplied keyword and date, and must come with images. Note that we cannot provide a formal definition for goal G1.3.1, since it is soft. Instead, the satisfaction of this goal can be inferred from its incoming contribution links, by performing an arithmetic mean on the satisfaction of each contributing goal weighted by the value given to each contribution link.

Operationalization [14] is the process that allows one to (semi-automatically) infer the operations that “implement” goals, and thus in our work that partner services must provide. An operation is defined through name, input and output values, and pre- and post-conditions. Required preconditions (*ReqPre*) define when the operation can be executed. Triggering conditions (*TrigPre*) define how the operation is activated. Required post-conditions (*ReqPost*) define additional conditions that must be true after execution. Domain pre- (*DomPre*) and post-conditions (*DomPost*) define the effects of the operation on the domain. Table 2 shows the result of the operationalization applied to the case study (except for operation *Find Text Content*). For example, operation *Find Graphical Content* moves the system from a state in which a container for the news that have to be collected is initialized (*DomPre*) to a state

[2] LTL provides the following operators: *sometimes in the future* ($\Diamond$), *sometimes in the past* ($\blacklozenge$), *always in the future* ($\Box$), *always in the past* ($\blacksquare$), *always in the future until* (U), and *always in the past since* (S), *in the previous state* ($\bullet$), and in the next state ($\circ$).

in which a set of suitable news is available (*DomPost*). This operation is triggered as soon as the collection of news matching provided keyword and date is started (*TrigPre*). The effect of this operation is to collect a set of news that match the date and keyword provided by the user (*ReqPost*). The definition of operation *Find Text Content* is similar to operation *Find Graphical Content* except for the required post-condition that is specified as follows:

ReqPost: $\forall(n \in nc.news; n.keyword = nc.keyword \lor nc.date = n.date \land$
$n.text \neq null \land (\exists(i \in n.images; i.content \neq null);\ true) \land \exists(n \in nc.news;\ true))$

The goal model also specifies a set of agents able to perform one or more operations. According to our point of view, agents represent the providers of the services that will be used in the composition. For example, agent A1 is the user issuing the requests and to whom news must be shown. Agent A3 can find news in both text and graphical mode, while agent A2 can only find news in text mode.

2.2 Fuzzy goals

The definition of goals through LTL formulae allows one to assess whether a goal is satisfied, but there is no way to say if it is only satisfied partially. For example, the definition of goal G1.3.2 only allows one to assess whether the global response time does not exceed the maximum threshold (RT_{MAX}), but it provides no information about the distance between the actual value and RT_{MAX}. Furthermore the definition of goal G1.4 only specifies whether the number of servers is lower than a certain value N_{MAX}, but it says nothing about the actual number of servers used in the pool. These are only a couple of examples that made us introduce fuzzy goals, and express their satisfaction level through real numbers between 0 and 1.

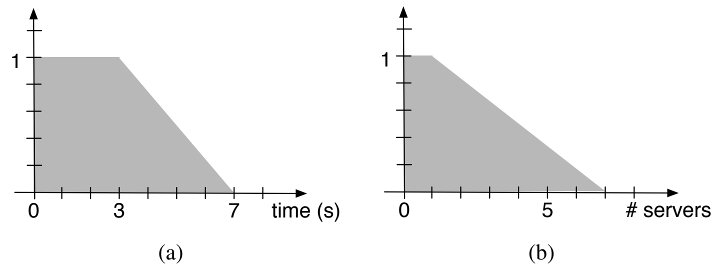

Fig. 2 Membership functions for goals G1.3.2 (a) and G1.4 (b).

Fuzzy goals are rendered through the operators already introduced in RELAX [27] to represent non-critical requirements: *AS EARLY/LATE AS POSSIBLE* ϕ, for temporal quantities, *AS CLOSE AS POSSIBLE TO* q ϕ, to assess the proximity of quantities or frequencies (ϕ) to a certain value (q), *AS MANY/FEW AS POSSIBLE* ϕ, for quantities (ϕ). This way goals G1.3.2 and G1.4 can be redefined in terms of these operators as follows:

G1.3.2 : *AS EARLY AS POSSIBLE t*

G1.4 : *AS FEW AS POSSIBLE servers*

Goal G1.3.2 now says that the response time t must be as short as possible, while goal G1.4 says that the number of servers must be as low as possible. The assessment of goals G1.3.2 and G1.4 is guided by the membership functions shown in Figure 2 that assign a satisfaction value between 0 and 1, depending on the actual response time (Figure 2(a)) and the number of used servers (Figure 2(b)), respectively. For example, as for goal G1.3.2 if the response time is less than 3 s, the satisfaction is 1, if the response time is between 3 s and 7 s the satisfaction has a value between 0 and 1, and if the response time is greater than 7 s the satisfaction is 0. Note that these functions are limited[3] and, in general, have a triangular or trapezoidal shape. The severity of membership functions can be measured in terms of the gradient of the inclined sides. The severity can be tuned according to the priority assigned to a goal (the higher the priority is, the steeper the membership function becomes).

2.3 Adaptation goals

Adaptation goals augment the KAOS model to describe and tune the adaptation capabilities associated with the system-to-be that are necessary to react to changes or to the low satisfaction of conventional goals. An adaptation goal defines a sequence of corrective actions to preserve the overall objective of the system. Each adaptation goal is associated with a *trigger* and a set of *condition*s. The trigger states when the adaptation goal must be activated. Conditions specify further necessary restrictions that must be true to allow the corresponding adaptation actions to be executed. Conditions may refer to properties of the system (e.g., satisfaction levels and priorities of other goals, or adaptation goals already performed) or domain assumptions.

Each adaptation goal is operationalized through adaptation *actions*.

- **Add, remove, or modify a conventional goal**;
- **Add, remove, or modify an adaptation goal**;
- **Add or remove an operation**;
- **Add or remove an entity**;
- **Perform an operation**, moves the process execution to the activity in which the operation, provided as parameter, starts to be performed (i.e., the first activity in the process flow associated with that operation);
- **Perform a goal**, moves the process execution to the activity in which the goal, provided as parameter, starts to be active (i.e., the first activity in the process flow associated with the first operation of the goal);

[3] Membership functions do not continue to be greater than 0 when the response time is infinite.

- **Substitute agent**.

Adaptation actions can be applied globally, on all (next/running) process instances, or locally (only on the application instance for which the triggers and conditions of that adpation goal are satisfied). Adaptation goals may also conflict when they are associated with conflicting goals (i.e., a couple of goals linked by a contribution link with a negative weight). In this case, we trigger the adaptation goal associated with the goal with the highest priority.

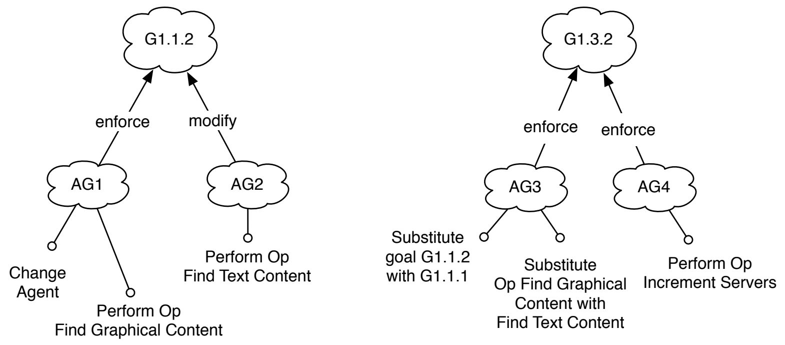
G1.1.2
enforce
modify
AG1
AG2
Change Agent
Perform Op Find Graphical Content
Perform Op Find Text Content
G1.3.2
enforce
enforce
AG3
AG4
Substitute goal G1.1.2 with G1.1.1
Substitute Op Find Graphical Content with Find Text Content
Perform Op Increment Servers

Fig. 3 Adaptive goals for the news provider.

The adaptation goals envisioned for our example are shown in Figure 3. Adaptation goals AG1 and AG2 are triggered when goal G1.1.2 is violated (i.e., its satisfaction is less than 1). AG1 is performed when the satisfaction of goal G1.1.2 is less than 0.7 and comprises two basic actions: it changes the agent that performs operation *Find Graphical Content* with another one (e.g., A5) and executes the same operation. These actions are applied locally, only for the instance of the goal model (and indeed, the process instance) for which the triggers and conditions hold true. The objective of this countermeasure is to enforce the satisfaction of goal G1.1.2. Adaptation goal AG2 is applied when the satisfaction of goal G1.1.2 is less than 0.7 and AG1 has been already applied. AG2 performs operation *Find Text Content*, and enforces a modified version of goal G1.1.2 (i.e., enforces goal G1.1.1 instead of G1.1.2). AG2 is also applied locally. Adaptation goals AG3 and AG4 are triggered when goal G1.3.2 is violated. In particular they are applied when the average value of the end-to-end response time of the news provider is greater than 3 s (conditions). AG3 enforces the satisfaction of goal G1.3.2 by switching to textual news (i.e., it substitutes goal G1.1.2 with goal G1.1.1 and operation *Find Graphical Content* with *Find Text Content*). AG3 is applied globally on all process instances. If AG3 is not able to reach its objective, AG4 is applied. Instead, it tries to enforce the satisfaction of goal G1.3.2, by incrementing the number of servers in the pool according to the severity of violation (it performs operation *Increment Servers*). Operation *Increment Servers* can only be performed by agent A4 and modifies the number of servers used by the load balancer. AG4 is also applied on all process instances. Adaptation goals

AG1 is in conflict with AG3 and AG4 since they try to enforce conflicting goals. According to our policy, AG3 and AG4 are triggered first, since they are associated with goal G1.3.2, which has higher priority ($p = 5$) than G1.1.2 ($p = 3$).

3 From Goals to Self-adaptive Compositions

This section illustrates our proposal to transform the goal model into running, self-adaptive service-oriented compositions. The operationalization of conventional goals is used to derive suitable compositions, while adaptation goals help deploy probes needed to collect enough data for the runtime evaluation of goals' satisfaction. They are also in charge of adaptation actions.

3.1 Runtime infrastructure

The runtime infrastructure works at two different levels of abstractions: process and goal level.

- The **process level** provides a BPEL engine to support the execution of the process instances. It also performs data collection and adaptation activities. Data collection activities gather the runtime data needed to update the state of entities, detect events, and evaluate the satisfaction of goals. Data to be collected can be internal (they belong to the process state), or external (they belong to the environment, and are retrieved by invoking external probes). Adaptation activities apply the actions associated with adaptation goals. Different probes and adaptation components can be easily plugged-in to obtain a complete execution platform.
- The **goal Level** keeps a live goal model for each process instance, and updates it by means of the data collected at process level. Every time an instance of the goal model is updated, the infrastructure recomputes the satisfaction of conventional goals. Specific analyzers can be plugged-in to when necessary, depending on the kind of constraint (i.e., LTL, FOL, fuzzy) that must be evaluated to assess a goal. The goal level also evaluates the triggers and conditions of the adaptation goals and decides when adaptation must be performed. Adaptation actions can affect both the goal model and the process instances. The interplay between the goal and process levels is supported in the infrastructure by a bidirectional mapping between the elements of the two levels.

Figure 4 shows the overall architecture of the runtime infrastructure. The *BPEL Engine* is an instance of ActiveBPEL Community Edition Engine [1] augmented with aspects [13] to collect internal data and start/stop the process' execution when necessary. The *Data Collector* coordinates the different probes, the *Adaptation Farm* oversees the activities of recovery components. The *Supervision Manager*,

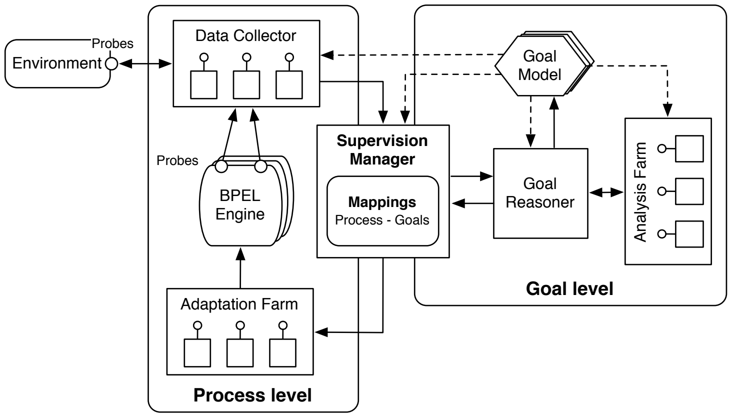

Fig. 4 Runtime infrastructure.

based on JBoss rule engine [22], receives data from the process level, and triggers the updates of the goal level. Also the *Goal Reasoner* is based on JBoss rule engine [22]: for each running process instance it keeps a goal model in its working memory and updates it. The *Goal Reasoner* asks the *Analysis Farm*, which coordintates analyzers, to (re-)compute the (degree of) satisfaction of the different leaf goals every time new data from the process level feed the goal model. The *Goal Reasoner* evaluates the triggers and conditions associated with adaptation goals and initiates their execution if needed. This means that the *Goal Reasoner* can modify the goal model and propagate the effects of adaptation at the process level. These effects are then applied onto the process instances by using the *Recovery Farm*, through the *Supervision Manager*.

3.2 Service Compositions

Service compositions are rendered as BPEL processes. Their activities, events, and partner services have a direct mapping onto the operations, entities, and agents of the goal model. Our assumption is that all operations associated with the same goal define a sequence and are not interleaved with the operations associated with other goals. The definition of a complete process requires the composition of these sequences and the transformation of their operations into the "corresponding" BPEL activities.

Each sequence is defined by encoding the operations associated with each goal in Alloy to check whether there exists a possible sequence of operations whose execution guarantees the satisfaction of the corresponding goal. Interested readers can refer to [19] for a complete presentation. In general, a sequence $s1$ can unconditionally precede $s2$ if the ending operation of $s1$, $op1$, and the starting operation of $s2$,

$op2$ satisfy equation 1. While a sequence $s1$ conditionally precedes $s2$ if the ending operation of $s1$ and the starting operation of $s2$, $op2$, satisfy equation 2.

$$(domPost(op1) \rightarrow domPre(op2)) \wedge (reqPost(op1) \rightarrow reqPost(op2) \wedge trigPre(op2)) \quad (1)$$

$$(domPost(op1) \rightarrow domPre(op2)) \wedge (trigPre(op1) \rightarrow trigPre(op2)) \wedge (reqPre(op2) \rightarrow reqPost(op1)) \quad (2)$$

In this last case an `if` activity is inserted in the BPEL process between $s1$ and $s2$, and its condition must correspond to the required precondition of $op2$.

For example, Figure 5(a) shows two possible sequences of operations. Since operation *Find Text Content* and *Find Graphical Content* are mutually exclusive, we select the first one to satisfy goal G1.1.2 ($p = 3$). This is because it is more critical than goal G1.1.1 ($p = 2$), which is associated with operation *Find Text Content*.

The generation of BPEL activities is semi-automatic. When an operation, in the goal domain, is translated into different sequences of BPEL activities, the user must select the most appropriate. Rules for translating operations into BPEL activities are the following:

1. If a required postcondition only contains an event, we generate one of the following activities: `invoke`, `invoke-receive`, or `reply`.
2. If the triggering precondition does not contain any event and the required postcondition changes some entities, we generate an `assign` for each change.
3. If the triggering precondition contains an event, it is translated into a `pick`. If the event refers to a temporal condition, it is translated into a `pick on_alarm`.
4. If rule 1 and 2 are true at the same time, we generate an `invoke-receive` followed by the set of `assign`s.
5. If rule 1 and 3 are verified at the same time, we generate an `invoke`, or a `pick`, or an `invoke-receive`.
6. If rule 2 and 3 are verified at the same time, we generate either a `pick` or a `receive`, and then a set of `assign`s.

The operations devised for the news provider are translated into the sequence of activities shown in Figure 5(b). Operations *Collect Requests* and *Find Graphical Content* follow rules 2 and 3. Since event *Collect* appears in the definition of both operations, `invoke News Provider` is generated only once. Operation *Provide Content*, instead, follows rule 1. The same applies also for those operations used in adaptation goals. For example, operation *Increment Servers* follows rule 1 and is simply translated into an `invoke`.

After identifying a proper sequence of BPEL activities, we must link entities and agents to proper process variables and partner links. Each entity used in the operationalization of conventional goals is rendered as an internal variable of the process. For example, our process has an internal variable called `NewsCollection`, which corresponds to entity *NewsCollection*. We also create a partner link for each agent and we assume that its endpoint reference is manually inserted by the user. In our example, we need partner services S1, S2, S3, S5 that match agents A1, A2, and A3, A5 respectively. Furthermore, we map agent A4 to another service S4 in charge of modifying the number of adopted servers.

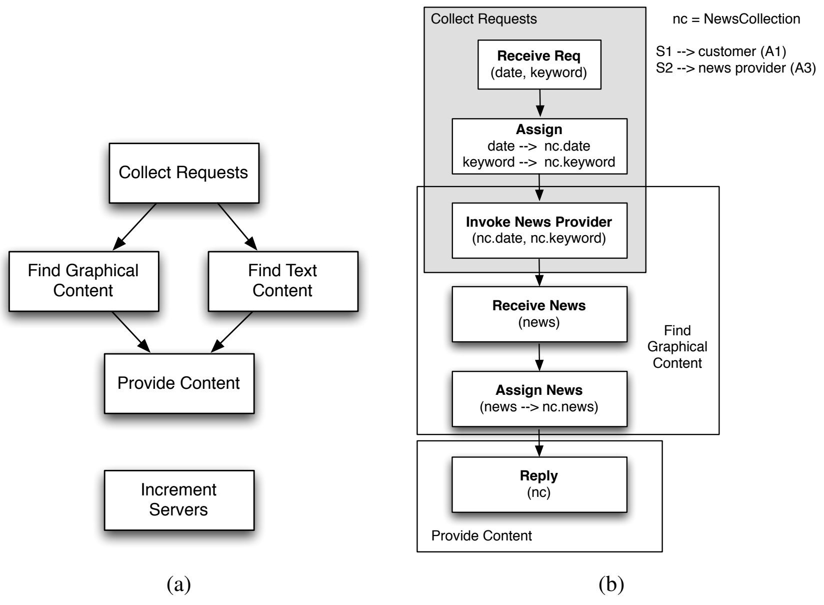

Fig. 5 (a) Two possible sequences of operations and (b) An abstract BPEL process.

3.3 Adaptation

This interplay between the process and goal levels is supported by the mapping of Table 3.

Conventional Goal (leaf)	XPath to the `sequence` in the BPEL process
Operation	- XPath to the first activity associated with the operation - XPath to the last activity associated with the operation
Agent	Partner link
Entitiy	Internal or external data
Event	XPath to a corresponding process activity
Adaptation goal	Recovery actions at process level

Table 3 Mapping goals to runtime data

Each conventional goal, which represents a functional requirement (i.e., it is operationalized), is mapped onto the corresponding `sequence` activity in the BPEL process (XPath expression). If the goal represents a non-functional requirement, but its nearest ancestor goal is operationalized, it is associated with the same `sequence` of its parent goal. The XPath expression provides the scope for both possible adaptation actions and for assessing the satisfaction of the goal (i.e., it defines the activities that must be probed to collect relevant data). Each operation is associated with the first and the last BPEL activities, associated with it through two XPath expressions. Each agent is associated with a partner service; the actual binding is manually inserted by the user. All events are mapped to an xpath pointing to the corresponding activity in the BPEL process. This activity must represent an interaction of the process with its partner services (e.g., `invoke`, `pick`, `receive`). Each adaptation goal is associated with a set of actions that must be performed at process level.

Data collection specifies the variables that must be collected at runtime to update the live instance of the goal model associated with the process instance. Data are collected by a set of probes that mainly differ on how (push/pull mode), and when (periodically/when certain events take place) data must be collected. If data are collected in push mode, the *Supervision Manager* just receives them from the corresponding probes, while if they are collected in pull mode, the *Supervision Manager* must activate the collection (periodically or at specific execution points) through dedicated rules.

To evaluate the degree of satisfaction of each goal, its formal definition must be properly translated to be evaluated by the selected analyzer. The infrastructure provides analyzers for FOL and LTL expressions, for crisp goals, and also provide analyzers to evaluate the actual satisfaction level of fuzzy goals. To this end, we built on our previous work and exploit the monitoring components provided by ALBERT [3], for LTL expressions, and Dynamo [4] for both FOL expressions and fuzzy membership functions.

To enact adaptation goals at runtime, the *Goal Reasoner* evaluates a set of rules on the live instances of the goal model available in its working memory. Each adaptation goal is associated with three kinds of JBoss rules. A **triggering rule**, activates the evaluation of the trigger associated with the goal. A **condition rule** evaluates the conditions linked to the goal. If the two previous rules provide positive feedback, an **activation rule** is in charge of the actual execution of the adaptation actions. performed when an adaptation goal can potentially fire (i.e., the corresponding Activation fact is available in the working memory) and is selected by the rule engine to be performed, among the other adaptation goals that can be performed as well. It executes the actions associated with that adaptation goal. For example, the triggering rule associated with AG1 is the following:

```
when
 Goal(id=="G1.1.2", satisfaction < 1, $pid: pID)
then
 wm.insert(new Trigger("TrigAG1", pid));
```

It is activated when the satisfaction of goal G1.1.2 is less than 1. This rule inserts a new `Trigger` fact in the working memory of the Goal Reasoner, indicating that the trigger associated with adaptation goal AG1 is satisfied for process instance `pid`. The corresponding condition rule is:

```
when
 $t: Trigger(name == "TrigAG1", $pid: pID)
 Goal(id=="G1.1.2", satisfaction < 0.7, pID == pid)
 $adGoal: AdaptationGoal(name=="AG1",
   $maxNumAct: maxAct, numOfActivations < $maxNumNAct)
then
 wm.remove($t);
 wm.insert(new Activation("AG1", pid));
```

It is activated when the condition associated with AG1 (the satisfaction of goal G1.1.2 is less than 0.7) is satisfied, the trigger of AG1 has taken place, and AG1 has been tried less than a maximum number of times (`maxNumAct`). It inserts a new fact in the working memory (`Activation`), to assert that the adaptation actions associated with goal AG1, for the process instance and the goal model corresponding to `pid` can be performed[4]. The action rule is:

```
salience 3
activation-group recovery
when
 $a: Activation(name == "AG1", $pid: pID)
 $ag: AdaptationGoal(name=="AG1", pID == pid)
then
 List<Action> actions = new ArrayList<Action>();
 actions.add(new SubstituteAgent("A3","A5"));
 actions.add(new Perform("Find Graphical Content");
 ag.numOfActivations++;
 Adaptation adapt =
    new Adaptation("AG1", actions,"instance", pid);
 adapt.perform();
 wm.remove(a);
```

Action rules have a priority (`salience`) equal to that of the goal they refer to (G1.1.2, in this case) and are always associated with activation-group `recovery`. This means that, once the rule with the highest priority fires, it automatically cancels the execution of the other adaptation goals that could be performed at the same time. Adaptation actions are performed when the triggers and conditions of the adaptation goal are satisfied (e.g., the corresponding activation object (`a`) is asserted in the working memory). The example rule performs the adaptation actions (`adapt.perform()`) on process instance (`pid`). Finally, it removes the object (`a`) that activated this adaptation.

[4] Note that if an adaptation goal is applied globally, there is no need to identify the process instance on which adaptation must be performed.

Adaptation actions associated with AG1 have no consequences on the goal model since they only require that the process re-execute the activities associated with operation *Find Graphical Content* by using another agent. At the process level these actions are applied locally and substitute partner service S1 with another one and restore the execution to operation *Find Graphical Content*. This is achieved through a Dynamo recovery directive that configures AOP probes to intercept the process execution before activity `invoke News Provider` and invoke operation *rebind(S3, S2.wsdl)*, that takes in input the name of the partner service to be substituted and the wsdl exposed by the new partner service to be adopted. After this operation is performed, the execution can proceed. This is only feasible with stateless services: in general, the application of an adaptation action cannot compromise the internal state of the process and that of its partner services.

If we consider adaptation goal AG3, it is applied globally and substitutes goal G1.1.2 and operation *Find Graphical Content* with goal G1.1.1 and operation *Find Text Content*, respectively. To this aim, we deploy a new version of the process, shown in Figure 3.3, for the next process instances. To apply AG3 on the running process instances we intercept the process execution just before activity `invoke` News Provider is performed. If a process instance has overtaken this execution point, it cannot be migrated. At this point, we substitute the activities associated with operation *Find Graphical Content* with the activities of the alternative execution path, shown in Figure 3.3. Then, the process execution proceeds, performing the activities of the alternative execution path.

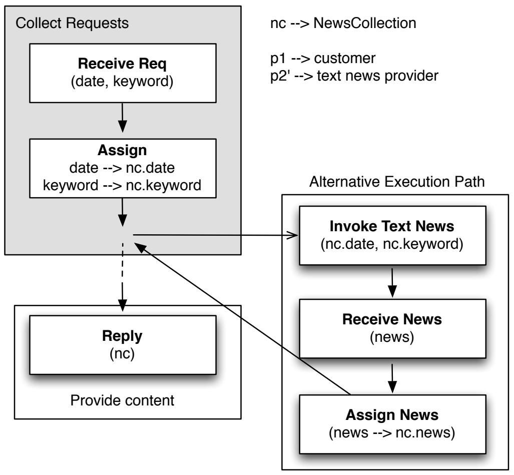

Fig. 6 Adapted process for Z.com.

4 Preliminary Validation

The validity of the proposed goal model has been evaluated by representing some example applications commonly used by other approaches proposed to model self-adaptive systems: an intelligent laundry [6], a book itinerary management system [23], and a garbage cleaner [17]. These experiments said that our goal model proved to be expressive enough to represent the main functionality of these systems together with their adaptation scenarios.

In the first case study, a laundry system must distribute assignments to the available washing machines and activate their execution. The system must also guarantee a set of fuzzy requirements stating that the energy consumed must not exceed a maximum allowed and the number of clothes that have to be washed must be low. These requirements are fuzzy since their satisfaction depends on the number of clothes to be washed and the amount of energy consumed, respectively. The satisfaction level of the energy consumed allows us to tune the duration of the washing programs accordingly. The adaptation goals devised for this case study also allow us to detect transient failures (e.g. the washing machine turns off suddenly) and activate an action that performs an operation to restart a washing cycle.

The itinerary booking system must help business travellers book their travels and receive updated notifications about the travel status (e.g., delays, cancelled flights). These notifications can be sent via email or SMS depending on the device the customer is actually using (i.e., laptop or mobile phone). Since sending an SMS is the most convenient option, we decided to adopt it in the base goal model of this case study. Suitable adaptation goals allow us to detect, through a trigger (i.e., whether the mobile phone is turned off) and a condition (i.e., whether the email of the customer's secretary is part of the information provided by the customer), when the customer's mobile phone is turned off, and apply an adaptation action that sends an email to him/her.

In the cleaner agent scenario, each agent is equipped with a sensor to detect the presence of dust and its driving direction. In case an agent finds a dirty cell, it must clean it, putting the dust in its embedded dustbox. The adaptation goals envisioned for this example allow the cleaner agent to recharge its battery when the load level is low. Furthermore, they allow us to cover a set of failure prevention scenarios. For example, adaptation goals can detect known symptoms of battery degeneration (e.g., suddenly reduced lifetime or voltage) and perform an operation to alert a technician, or get a new battery. Adaptation goals can also detect the presence of obstacles in the driving direction of an agent and activate two actions: stop the agent and change the driving direction, when possible.

These exercises demonstrated to be very useful to highlight both the advantages and disadvantages of our approach. We can perform accurate and precise adaptations by assessing the satisfaction degree of soft goals and tuning the adaptation parameters accordingly, as described before. The usage of triggers and conditions makes it possible to react after system failures or context changes, and also model preventive adaptations to avoid a failure when known symptoms take place.

We adopt a priority based mechanism to solve conflicts among adaptations that can be triggered at the same time. This mechanism is still too simplistic in certain situations. For example a vicious cycle may exist when a countermeasure A has a negative side effect on another goal, and that goal's countermeasure B has a negative side effect on the first goal as well. These cases can be handled by tuning the conditions of the countermeasures involved, which would become pretty complex. For this reason, other decision making mechanisms should be adopted, like trade-off optimization functions. Finally our goal model does not provide any reasoning mechanism to automatically detect possible adaptations in advance, after changes in the context and in the stakeholders' requirements take place.

5 Related Work

Our proposal aims to provide a goal-based methodology to model the requirements of service compositions, that is, the architecture of service-based applications. Cheng et al. [7] proposed a similar approach for self-adaptive systems in general. The authors detect the reasons (threats) that may cause uncertainty in the satisfaction of goals, and propose 3 strategies for their mitigation: add new functionality, tolerate uncertainty, or switch to a new goal model that is supposed to repair the violation. Instead our strategies do not constraint the ways a goal model can be modified, but they can have different objectives and severity. These features allow us to solve conflicts among strategies and provide ways to apply them at runtime. Also Goldsby et al. [10] use goal models to represent the non-adaptive behavior of the system (business logic), the adaptation strategies (to handle environmental changes) and the mechanisms needed by the underlying infrastructure to perform adaptation. These proposals only handle adaptation by enumerating all alternative paths at design time. In contrast, we support the continuous evolution of the goal model by keeping a live goal model for each process instance and by modifying it at runtime.

Different works have already tried to link service compositions with the business objectives they have to achieve. For example, Kazhamiakin et al. [12] adopt Tropos to specify the objectives of the different actors involved in a choreography. Tropos tasks are refined into message exchanges, suitable annotations are added to express conditions on the creation and fulfillment of goals, and assume/guarantee conditions are added to the tasks delegated to partner services. These elements enable the generation of annotated BPEL processes. These processes can only be verified statically through model checking, ours also embed self-adaptation capabilities.

Another similar approach is the one proposed by Mahfouz et al. [15], which models the goals of each actor and also the dependences among them. Actor dependencies take place when a task performed by an actor depends on another task performed by a different actor. Dependencies are then translated into message sequences exchanged between actors, and objectives into sets of local activities performed in each actor's domain. The authors also propose a methodology to modify a choreography according to changes in the business needs (dependencies between actors and local objectives). Although this approach traces changes at requirements level, it does not provide explicit policies to apply these changes at runtime.

The idea of monitoring requirements was originally proposed by Fickas et al. [9]. The authors adopt a manual approach to derive monitors able to verify requirements' satisfaction at runtime. Wang et al. [26] use the generation of log data to infer the denial of requirements and detect problematic components. Diagnosis is inferred automatically after stating explicitly what requirements can fail. Robinson [24] distinguishes between the design-time model, where business goals and their possible obstacles are defined, and the runtime model, where logical monitors are automatically derived from the obstacles and are applied onto the running system. This approach requires that diagnostic formulae be generated manually from obstacle analysis.

Despite a lot of work focused on monitoring requirements, only few of them provide reconciliation mechanisms when requirements are violated. Wang et al. [26] generate system reconfigurations guided by OR-refinements of goals. They choose the configuration that contributes most positively to the non-functional requirements of the system and also has the lowest impact on the current configuration.

To ensure the continuous satisfaction of requirements, one needs to adapt the specification of the system-to-be according to changes in the context. This idea was originally proposed by Salifu et al. [25] and was extensively exploited in different works [20] [2] that handled context variability through the explicit modeling of alternatives. Penserini et al. [20] model the availability of execution plans to achieve a goal (called ability), and the set of pre-conditions and context-conditions that can trigger those plans (called opportunities). Dalpiaz et al. [2] explicitly detect the parameters coming from the external environment (context) that stimulate the need for changing the system's behavior. These changes are represented in terms of alternative execution plans. Moreover the authors also provide precise mechanisms to monitor the context. All these works are interesting since they address adaptation at requirements level, but they mainly target context-aware applications and adaptation. They do not consider adaptations that may be required by the system itself because some goals cannot be satisfied anymore, or new goals are added. We also foresee a wider set of adaptation strategies and provide smarter mechanisms to solve conflicts among different strategies.

Despite our solution is more tailored to service-based applications, many works [11, 16] focus on multi agent systems (MAS). Morandini et al. [16], like us, start from a goal model, Tropos4AS [17], which enriches TROPOS with soft goals, environment entities, conditions relating entities and state transitions, and undesired error states. The goal model is adopted to implement the alternative system behaviors that can be selected given some context conditions. Huhns et al. [11] exploit agents to support software redundancy, in terms of different implementations, and provide software adaptation. The advantage here is that agents can be added/removed dynamically; this way, the software system can be customized at runtime and become more robust.

The main advantage of these agent-based systems is their flexibility, since adaptation actions are applied at the level of each single component. On the other hand, MAS provide no guarantees that agents cannot perform conflicting actions or that the main system's objectives are always achieved. Our approach, instead, is centralized and declare adaptation actions at the level of the whole system. Adaptation is simply achieved by adding, removing, and substituting components, since the SOA paradigm does not allow us to change the internal behavior of a component.

6 Conclusions

This chapter proposes an innovative approach to specify adaptive service-oriented architectures/applications. The proposal extends the KAOS goal model and accommodates both conventional (functional and non-functional) requirements and the requirements on how the system is supposed to adapt itself at runtime. Goals can be *crisp*, when their satisfiability is boolean, *fuzzy*, when they can also be partially satisfied, and related to *adaptation*, when they specify adaptation policies.

The proposal also explains how to map the "comprehensive" goal model onto the underlying architecture. Conventional goals are used to identify the best service composition that fits stated requirements. Adaptation goals are translated in data collection directives and sequences of concrete adaptation actions. The first assessment provided positive and interesting results.

We are already working on extending the tool support and on adopting our proposal to model other self-adaptive service compositions.

References

1. Active Endpoints: The ActiveBPEL Engine. `http://www.activevos.com/community-open-source.php`
2. Ali, R., Dalpiaz, F., Giorgini, P.: A Goal Modeling Framework for Self-Contextualizable Software. In: Proceedings of the 14th International Conference on Exploring Modeling Methods in Systems Analysis and Design, vol. 29, pp. 326–338 (2009)
3. Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., Spoletini, P.: Validation of Web Service Compositions. IET Softw. **1**(6), 219–232 (2007)
4. Baresi, L., Guinea, S.: Self-supervising BPEL Processes. IEEE Transactions on Software Engineering **99**(PrePrints) (2010)
5. Baresi, L., Guinea, S., Pasquale, L.: Integrated and Composable Supervision of BPEL Processes. In: Proc. of the 6th Int. Conf. of Service Oriented Computing, pp. 614–619 (2008)
6. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy Goals for Requirements-driven Adaptation. In: Proceedings of the 18th International Requirements Engineering Conference, pp. 125–134. IEEE Computer Society (2010)
7. Cheng, B.H.C., Sawyer, P., Bencomo, N., Whittle, J.: A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems, pp. 468–483 (2009)

8. Cheng, S.W., Garlan, D., Schmerl, B.: Architecture-based Self-adaptation in the Presence of Multiple Objectives. In: Proceedings of the 2nd International Workshop on Self-adaptation and Self-managing Systems, pp. 2–8. ACM (2006)
9. Fickas, S., Feather, M.S.: Requirements Monitoring in Dynamic Environments. In: Proceedings of the 2nd Inetrnational Symposium on Requirements Engineering, p. 140. IEEE Computer Society (1995)
10. Goldsby, H.J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Hughes, D.: Goal-Based Modeling of Dynamically Adaptive System Requirements. In: Proceedings of the 15th International Conference on Engineering of Computer-Based Systems, pp. 36–45 (2008)
11. Huhns, M.N., Holderfield, V.T., Gutierrez, R.L.Z.: Robust Software Via Agent-Based Redundancy. In: Proceedings of the 2nd International Joint Conference on Autonomous Agents & Multiagent Systems, pp. 1018–1019. ACM (2003)
12. Kazhamiakin, R., Pistore, M., Roveri, M.: A Framework for Integrating Business Processes and Business Requirements. In: Proceedings of the 8th International Con on Enterprise Distributed Object Computing, pp. 9–20. IEEE Computer Society (2004)
13. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J., Irwin, J.: Aspect-Oriented Programming. In: Proceedings of the 11th European Conference on Object-Oriented Programming, pp. 220–242. Springer (1997)
14. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. John Wiley (2009)
15. Mahfouz, A., Barroca, L., Laney, R.C., Nuseibeh, B.: Requirements-Driven Collaborative Choreography Customization. In: Proceedings of the 7th International Joint Conference ICSOC-ServiceWave, pp. 144–158 (2009)
16. Morandini, M., Penserini, L., Perini, A.: Modelling Self-Adaptivity: A Goal-Oriented Approach. In: Proceedings of the 2nd International Conference on Self-Adaptive and Self-Organising Systems, pp. 469–470. IEEE Computer Society (2008)
17. Morandini, M., Penserini, L., Perini, A.: Towards Goal-Oriented Development of Self-Adaptive Systems. In: Proceedings of the 3rd International Workshop on Software Engineering for Adaptive and Self-Managing Systems, pp. 9–16. ACM (2008)
18. OASIS: Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html` (2007)
19. Pasquale, L.: A Goal-oriented Methodology for Self-supervised Service Compositions. Ph.D. thesis, Politecnico di Milano (2011)
20. Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: High Variability Design for Software Agents: Extending Tropos. ACM Transanctions Autonomous Adaptive Systems **2**(4), 75–102 (2007)
21. Pnueli, A.: The Temporal Logic of Programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. Weizmann Science Press of Israel (1977)
22. Proctor, M., et al.: Drools. `http://www.jboss.org/drools/`
23. Qureshi, N.A., Perini, A.: Engineering Adaptive Requirements. In: Proceedings of the 4th International Workshop on Software Engineering for Adaptive and Self-Managing Systems, pp. 126–131. ACM (2009)
24. Robinson, W.N.: Monitoring Web Service Requirements. In: Proc. of the 11th Int. Requirements Engineering Conference, pp. 65–74 (2003)
25. Salifu, M., Yu, Y., Nuseibeh, B.: Specifying Monitoring and Switching Problems in Context. In: Proc. of the 15th Int. Requirements Engineering Conference, pp. 211–220. IEEE (2007)
26. Wang, Y. and Mylopoulos, J.: Self-repair Through Reconfiguration: A Requirements Engineering Approach. In: Proceedings of the 24st International Conference on Automated Software Engineering, pp. 257–268. IEEE Computer Society (2009)
27. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C.: RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In: Proceedings of the 17th International Requirements Engineering Conference, pp. 79–88. IEEE Computer Society (2009)