An Agent-based Approach to Composing Web Services to Support Adaptable Business Processes

Hong Mei, Xi Sun, Xuanzhe Liu, Wenpin Jiao, Gang Huang

Institute of Software, School of Electronics Engineering and Computer Science
Peking University, Beijing 100871, China

{meih, sunxi, liuxzh, jwp, huanggang}@cs.pku.edu.cn

Abstract: Business processes built from web services need a more adaptable composition solution. In this paper, an approach based on agents is proposed to control the executions of business processes via agent behavior rules, which can be generated automatically and modified dynamically to enable the adaptations of business process. In the approach, the adaptations of the business process are specified in independent adaptation units and agents can load and interpret user-defined adaptation units at runtime. Thus, the executions of business processes can be adapted dynamically. This paper also describes a running support of lightweight agents on a reflective middleware, on which agents can be generated automatically to compose web service to support adaptable business processes according to the specifications of business processes and the adaptation units.

Keywords: Web Service, Process, Adaptation, Agent

1. Introduction

Web services are becoming one of the most important paradigms for the interoperability and integrations of distributed software systems on the Internet. Because of the platform and language independent presentations and loose-coupling message-based communications, web services have great potential in streamlining business-to-business or enterprise application integrations.

Web service composition lets developers create applications on top of service-oriented computing’s native
description, discovery, and communication capabilities. Many researches and practices have focused on this area to combine existing web services on the Internet seamlessly and accelerate rapid application development, service reuse, and complex service consummation [18].

Comparing with traditional workflow technologies, business processes built from web services need a more adaptable composition solution. The Internet is a highly dynamic environment, that is, service providers may enter or leave the network at anytime, and besides, under the business pressure, organizations involved in a web service process may often change their business policies, their partners, and their collaboration conditions. Therefore, the support for the adaptation of business processes has become an important issue in the web service composition research.

Generally, approaches to composing web services can be divided into two broad categories [25] but neither of the two categories of approaches is adaptable enough. The syntactic approach standardized by industry companies, for example, BPEL4WS [8], basically describes the order in which messages are exchanged among web services, which are specified by the signature and direction of the operations using WSDL [28]. Although having gained considerable acceptance in the web service community, this approach has some obvious limitations, for instance, it often assumes that the composite process of web services is predefined and does not evolve, which implies the whole process may crash if some specified service become unavailable, and moreover, modifications made to the process will result in restarting the whole process, which may be intolerable in some mission-critical areas such as e-Banking applications.

An alternative approach is developed by the semantic community using pre-agreed ontology which defines the input, output, precondition and effect (IOPE) of web services. OWL-S [21] is a service ontology that describes web services in a machine-interpretable way, which provides possibilities for automatic service discovery, selection, composition and execution (e.g., [16][22][24]). Based on semantic web services, much research work related to the automatic composition of web services has been done, for instance, [32][17]
proposed an approach to composing web services automatically based on artificial intelligent planning systems. However, this category of research work often assumes the composition is carried out in some static and closed environment rather than an open and changing environment like the Internet and it still needs further development and evaluation in real large-scale applications.

In the open and dynamic Internet-based environment, requested web services may become unavailable unpredictably due to the instability of the network or the dynamic entering or leaving of the service providers. In that case, to guarantee the availability of the business process, the system supporting the business process should be able to find alternative web services and replace those unusable web services online dynamically. However, though the same service may be implemented and provided by many service providers, the ways of providing the service are very probably varied on many aspects, for instance, the interfaces and the supported interaction protocols. That is, when an alternative web service is selected to replace an unusable web service online, there may be mismatching interactions between the selected web service and the interaction environment supposed by the replaced web service [33][15]. Therefore, when an unavailable service is substituted, the system should be able to eliminate the mismatching interactions automatically.

On the other hand, the business decision policies controlling the execution of the business process may change over time for coping with different organizational requirements or different service-providing strategies, for instance, realizing different qualities of services to satisfy the requirements of different customers. However, current processes usually integrate business constraints and policies explicitly in the specifications. Such processes are not modular and they are complex and hard to maintain, which will make processes poorly reusable since all the business logic is defined in one unit and no part of it is easy to be reused [6].

Therefore, to build adaptable business processes based on web services on the Internet, the following issues should be well addressed at least.

1). When a specified web service in the process is unavailable, how to find an appropriate alternative web
service(s) to replace the failed web service dynamically without interrupting the execution of the process.

2). When a web service is selected to substitute a failed web service, how to guarantee the selected web service to be integrated seamlessly and interoperate well in the process.

3). When the business logic changes, how to adapt the execution of the process to the changes at runtime.

To achieve this, means of specifying the business logic separately from the process specification should be provided. By using the provided means, the business constraints or policies implementing the business logic could be dynamically loaded and activated.

In this paper, a framework based on software agents and business rules is proposed to add adaptation capabilities to business processes specified in the BPEL4WS. This work focuses on how to build adaptive business processes based on BPEL4WS since BPEL4WS is the de facto industrial standard and has been applied in the varieties of areas (e.g., finance, accounting, supply chain, manufacturing and etc.).

A software agent is defined as a computer system situated in an environment and capable of flexible autonomous actions in order to meet its design objectives [19][30]. Agent-based systems are well-positioned to enable intelligent process integration as they potentially provide some of the basic infrastructure capabilities. For addressing the aforementioned issues well, the business process should be monitored at runtime to adapt to the changes of the environment dynamically. Software agent is a natural solution because it can sense the environment at runtime and use the perceived information to evaluate and change its behaviors.

The framework uses agent-based behavior rules to control the executions of business processes so that the adaptation of business processes can be supported via modifying the behavior rules of agents. To guarantee these behavior rules of agents exactly as specified by the business process specifications, several algorithms are implemented to translate business process specifications into agent behavior rules equivalently.

However, the adaptation of a business process is often not foreknown or predictable since the business policies and the running environment may change timely. It is unpractical to modify the behavior rules of agents
manually when changes happen. Therefore, in the approach proposed in this paper, a mechanism is provided for people to define new business policies to meet new business requirements or new strategies to cope with changes of the environment. Thus, people can define new adaptation policies as pluggable units independently of the process specification and then agents will dynamically load the units and automatically translate them into adaptation rules.

The proposed framework’s advantages can be presented as the following features:

♦ **Using the behavior rules of agents to control the execution of the process.** In the framework, the business policies can also be specified as behavior rules. Thus, on the one hand, the business policies can be specified separately from the process specification. On the other hand, after the process specification is translated into equivalent behavior rules, agents will handle the business logic and the process specification in a uniform framework, which will enable to integrate them together seamlessly. Moreover, the adaptations of business processes can be easily realized via modifying business rules at runtime.

♦ **Supporting Context-aware process adaptations.** The framework is grounded on software agents to collect runtime environment information and activate rules to control the executions of processes. Software agents can be considered as autonomous computational entities capable of operating efficiently in dynamic and open environments, and using adaptation rules, agents can make decisions and adjust their behaviors according to the perceived environment information, for example, the service status, the network bandwidth, etc.

♦ **Providing a flexible mechanism for future extension.** Several “adaptation units” are developed to facilitate adding adaptation strategies into existing web service processes. Such units can be specified independently of the process specification and they are pluggable so that new units can easily be added for extending the framework’s ability.

The remainder of this paper is organized as follows. Section 2 introduces related work. Section 3 describes the agent-based framework to support adaptable processes, including the process specification transformation
algorithm. Section 4 describes the implementation of the framework and gives an example to demonstrate the usage of the framework; and Section 5 concludes this paper and discusses the future work.

2. Related Work

The WSDL is the specification to define a collection of the interfaces of atomic web services, which describes how to access to the services rather than what happens from invoking the operations. Therefore, many researches leverage techniques from the semantic community to describe Web service capabilities, interaction patterns, and domains along with a logic that allows reasoning on the information [24]. Maximilien and Singh [16] address dynamic service selection via an agent framework coupled with a QoS ontology. This approach complements OWL-S by emphasizing the quality aspects. Although these semantic-based approaches provide flexible ways to compose services automatically, further studies are still necessary to evaluate the usability and scalability in real applications.

The Business Rules Group defines a business rule as a statement that defines or constrains some aspect of the business. The business rule approach encompasses a collection of terms (definitions), facts (connection between terms) that contain sensible business relevant observations, and rules (computation, constraints and conditional logic) that used to discover new information or guide decision-making [4]. Orriens et al. [20] presents a rule driven mechanism to govern and guide the process of service composition in terms of five composition phases spanning abstract definition, scheduling, construction, execution and evolution to support on-the-fly business process generation. This approach does not address the issue about potential mismatches in retrieving and composing activities, and rule conflict resolution is not discussed, either.

Though rule is a flexible and natural representation to model the business logic of a composite process, BPEL4WS, as a representative example of process-based languages, fails to provide the support for explicitly representing the declarative rules in a clean and decoupled way. Researches [6] [7] [26] focused on the applicability of AOP (Aspect Oriented Programming) for integrating business rules into business rules; in these
approaches, rules are modularized as *advices* specifying some crosscutting concerns and weaved into the process specification at certain *joint points*. However, in this category of approaches, the inferring and conflict management of rules introduced by the advices completely relies on programmers and the maintenance of the dependencies among rules is a tedious and tough work for programmers.

Agent-based approaches for business process management have been suggested in the ADEPT system [1] and the DESIRE framework [3]. Many of these approaches, attempt at extending the applicability of agent-based technologies to organizational process management. The applications have been restricted to stable organizational processes such order processing and inventory management, wherein, the underlying business knowledge does not change frequently. Additionally, the utility of these systems in supporting ad hoc business processes such as engineering design has yet to be established. The complexity of capturing the requisite design process and product-related knowledge makes the task onerous.

Ermolayev [10] presents a framework for agent-enabled dynamic web service composition. The core of the methodology is to consider web service as an agent capability having proper ontological description, and a Middle Agent Layer is introduced to conduct service request to task transformation, decomposition and performance. This approach focus on issues about task delegation and negotiation between agents, and little work has been done to adapt process logic dynamically to meet business requirement change.

Vidal et al. [27] describes an approach to build adaptive multiagent-based workflow systems. The authors develop several tools for mapping BPEL4WS workflows into Petri nets that they can use to generate multiagent instantiation of the workflow. The algorithms for effective similarity matching and contextual substitutions are developed to let agents intelligently diverge from prescribed workflows when needed. However, Petri nets are not a very scalable modeling technique, and besides, in this approach, business policies of process are embodied into the model and hard to maintain.

However, the existing approaches usually can operate well on only one or two aspects, such as supporting
the business process by using rules or agents (e.g., [20][1][3]), allowing to discover, select and replace web services dynamically (e.g., [16][10][27][4]), or enabling to define the business logic independently of the process specification and inject the business policies into the execution of the process by using AOP technology at runtime (e.g., [6][7][26]). Therefore, a better approach should be capable of doing well on all of those aspects, such as supporting the dynamic replacements of unavailable web services, guaranteeing the interoperability of the alternative web services in the process, enabling runtime defining and loading adaptation business policies, and facilitating future extensions of adaptable processes. In the next section, an agent-based framework addressing these issues will be described in detail.

3. Agent-Based Web Service Composition Framework

In this section, an architectural overview about the agent-based web service composition framework is given first, the algorithms for extracting the business rules from the BPEL4WS specification is described then, the equivalence between the specification and the rule set is proven next, and how to extend the rule set for adaptability using adaptation units is discussed at last.

3.1. Architectural Overview

Figure 1 shows the agent-based adaptable business process framework from the architectural perspective.

![Figure 1. The agent-based web-service composition framework](image-url)
The input needed by the agent is the BPEL4WS process specification and adaptation units. An adaptation unit encapsulates the process definer’s adaptation strategy, for example, to introduce a service as a “backup” for some service in the process, which will be explained in detail in section 3.3.

The Interpreter translates the specification into the behavior rules of the agent, and these rules will be used to manage the process execution at runtime.

The agent uses sensors to receive the messages sent by other web services and monitors the runtime environment. The information will be converted into knowledge objects that can be accepted by the built-in rule engine to activate aforementioned rules. The Behavior manager is responsible for the execution of the agent’s behaviors, which are pre-defined in the behavior library. The behaviors include using effecters to invoke certain services, updating internal variables according to the respondent messages, selecting services from a group of candidates based on the adaptation policies, and so on.

3.2. Transforming the BPEL4WS Process into Agent Behavior Rules

In the framework, an agent will take the business process specification as the input to generate the behavior rules automatically and then trigger the behavior rules to realize the execution of the business process.

The execution of a business process can be controlled by two categories of behavior rules of agents.

- **Activity rules**: Activity rules govern when and how activities are to be performed in the composition.
  
  For example, for an individual activity, the behavior rule for activating its execution can be specified as follows.

  **Condition**: The activity is the current activity that the process is trying to perform, and It is not blocked for synchronizing.

  **Action**: Perform the latter activity.

  For two activities wrapped by an `<sequence>` composite activity should be performed in the literal order, and the rule can look like:

  **Condition**: The former activity finishes, or some anchor indicating the former’s finish appears.

  **Action**: Try to perform the latter activity.
For a composite activity such as `<while>` or `<switch>` with entrance condition(s), the corresponding rules can be like:

**Condition**: The process is trying to perform the composite activity and the entrance condition(s) are satisfied currently.

**Action**: Enter the activity block and try to perform those activities involved in the block.

**Condition**: Otherwise

**Action**: Leave the activity block unperformed.

Generally speaking, the compensation and error handling activities in BPEL can be considered as a kind of process adaptation. In the framework, adaptation units are introduced in section 3.3 to specify these adaptation strategies. So when the process specification is parsed, these activities will be converted into adaptation units manually instead of activity flow rules. In the future, how to automate the translation process will be investigated.

- **Dependency rules**: Dependency rules represent the dependencies between activities. An typical kind of dependency is *synchronization dependency* described by the `<link>` construct of BPEL4WS specification, which means an activity does not start until the status of all its incoming links has been determined (For detail, see [8]). Corresponding rules can be specified as follows:

  **Condition**: The target activity is blocked in synchronization, and All of the source activities have finished.

  **Action**: Try to perform the target activity.

Because dependency rules can be obtained easily via analyzing the `<link>` constructs in the process specification, this work will only discuss how to extract activity rules from the process specification.

### 3.2.1. Extracting Activity Rules from the Process Specification

For simplifying the description of the algorithms for translating the BPEL4WS-based business process specification into the behavior rules, an activity involved in the business process is be specified as a quintuplet.

\[ \text{Activity} = <id, \text{start}, \text{finish}, \text{next}, \text{parent}> \]

Here *id* is the unique identity (*i.e.*, the name if there is specified one) of the activity.
start is the starting address of the activity in the process whilst finish is the termination address. Because the execution of a process is much similar to the execution of a program on a computer and the position of an activity in a process is like the instruction counter of an instruction (or statement) in a program, the notion of address is used to refer to the position of an activity. In addition, a composite activity in a process includes a collection of other activities and its execution crosses several addresses, so the starting address and the termination address are used to indicate where the execution of the activity will start and terminate, respectively. For the first activity occurring in the process, the value of start is 1; whilst for the last activity, the value of finish is designated as –1.

Similarly as the next instruction counter, next is the position of the next activity literally succeeding the current activity. When the activity is an individual activity, the value of next is equal to that of finish; otherwise, the value of next will be the same as the value of start of the first activity involved in the composite activity.

Lastly, parent is the identity of the composite activity that is composed of the current activity and other activities. For those outmost activities of the process, the values of parent are the name of the process by default.

The behavior rules implementing the process can be generated via using the following algorithm. For each atomic activity type, its execution logic is implemented as a corresponding agent behavior, in which the syntax and semantic information of the activity is processed. In algorithm 1, the statement “Do(Activity)” in the rules means the agent will perform a behavior to execute the activity according to its type and update the process internal variables.

Algorithm 1: Transforming process into agent behavior rules.

1. Compute <id, start, finish, next, parent> for every activity in the process by using Algorithm 2.
2. Suppose Pid is the identity of the process and initially Pid = process.name, and CA is a variable denoting the current activity. They will be substituted by their actual values in the generated rules.
3. Scan the process and generate the behavior rules corresponding to every activity involved in the process. In the rules, cur_process and cur_position are internal variables of the rule engine of the agent to record the current process’s id and the position of the current activity in the process, respectively.
   a. For an individual activity, such as <receive>, <reply>, <invoke>, <assign>, <terminate>, <wait>, and <empty>, generate a rule as follows.

\[
\text{cur_process} = \text{Pid and cur_position} = \text{activity.start} \Rightarrow
\]
Do (activity); cur_position = activity.next

b. For a composite activity, such as <sequence>, <switch>, <while>, and <flow>, generate the following rule.

i. The rule corresponding to activity <sequence> or <flow> is generated as follows.

\[ \text{cur_process}= \text{Pid and cur_position} = \text{activity.start} \rightarrow \text{cur_position} = \text{activity.next} \]

ii. When the activity is <switch>, generate a rule for each case branch, including the branch of <otherwise>. Suppose the <switch> activity includes \( N+1 \) branches, the condition selecting the \( i \)th \( (1 \leq i \leq N) \) branch is \( C_i \) and the triggered activity will be \( \text{activity}_i \) when the \( i \)th branch is selected (for the <otherwise> branch, the corresponding activity is denoted as \( \text{activity}_{\text{other}} \)), then generate the following rule set for a <switch> activity as follows.

\[ \text{cur_process}= \text{Pid and current_position} = \text{activity.start and } C_i \rightarrow \text{cur_position} = \text{activity}_i.\text{start} \]
\[ \text{cur_process}= \text{Pid and current_position} = \text{activity.start and not } (C_1 \text{ and } C_2 \text{ and } \ldots \text{ and } C_n) \rightarrow \text{cur_position} = \text{activity}_{\text{other}}.\text{start} \]

iii. For an activity <while> with an iterative condition \( C_w \), generate the following two rules.

\[ \text{cur_process}= \text{Pid and current_position} = \text{activity.start and } C_w \rightarrow \text{cur_position} = \text{activity.next} \]
\[ \text{cur_process}= \text{Pid and current_position} = \text{activity.start and not } C_w \rightarrow \text{cur_position} = \text{activity.finish} \]

The quintuplets related to activities involved in the process can be computed as follows.

**Algorithm 2. Computing the positions of activities.**

1. Suppose variable \( \text{position} \) is used to record the position of an activity met during scanning the process specification. Initially, set \( \text{position} \) to be 1.

2. Scan the process specification and compute as follows for every activity occurring in the process specification.

   a. Suppose the activity met currently is \( \alpha \), then compute the position of \( \alpha \).

      i. Let \( \alpha.\text{start} = \text{position} \), \( \alpha.\text{next} = \text{position} + 1 \).

      ii. If \( \alpha \) is an activity of <terminate>, let \( \alpha.\text{finish} = -1 \).

      iii. If \( \alpha \) is an individual activity rather than <terminate>, let \( \alpha.\text{finish} = \text{position} + 1 \).

      iv. Otherwise, if \( \alpha \) is a composite activity, count the number of activities involved in the composite activity, suppose the number is \( C \), and then let \( \alpha.\text{finish} = \text{position} + C + 1 \).

   b. Adjust the computation results as follows.

      i. If \( \alpha \) is the last and outmost activity of the process, let \( \alpha.\text{next} = \alpha.\text{finish} = -1 \).

      ii. If \( \alpha \) is an offspring of a composite activity, i.e., \( \alpha \) is involved in the composition activity, then

         1. If \( \alpha \)'s parent activity is a <flow>, let \( \alpha.\text{start} = \alpha.\text{parent}.\text{start} + 1 \). Thus, when the process enters a <flow> activity, all offspring activities will be activated concurrently.

         2. If \( \alpha \)'s parent activity is a <while>, let \( \alpha.\text{start} = \alpha.\text{parent}.\text{start} \). Thus, after \( \alpha \) finishes, the control of the execution of the process will go back to the <while> activity and re-evaluate the condition of the <while> activity.

         3. Otherwise, if \( \alpha \) is the last offspring of its parent activity, let \( \alpha.\text{next} = \alpha.\text{parent}.\text{finish} \).

   c. Let \( \text{position} = \text{position} + 1 \), and start to handle the next activity.
3.2.2. Proof of the equivalence

The behavior of an agent is said to be equivalent to the execution of a business process if the behavior rules of the agent control the performance of the activities involved in the process in the same way specified by the business process specification. That contains two aspects of implications: 1) all of the performances of activities in the process are embodied in the executions of the behavior rules of the agents, i.e., what the process specification supposes will be realized via the executions of the behavior rules, and 2) the behavior rules do not do more than that the process specification has supposed.

**Theorem 1.** The behavior rule set generated based on a specific process specification by using the above two algorithms equivalently realizes the process.

**Proof.** Due to the space limitation, here just inductively proves that the performances of activities specified by the business process are embodied in the execution of the agent specified by the behavior rules.

1. For any two literally adjacent activities, the behavior rules can guarantee that the preceding activity will always be performed before the succeeding activity. Without losing the generality, suppose the two adjacent activities are individual activities, $\alpha_1$ and $\alpha_2$. Then when $\text{cur\_position}$ is equal to the starting address of $\alpha_1$, the behavior rule related to $\alpha_1$ will be triggered and then $\alpha_1$ will be activated. After the execution of $\alpha_1$ finishes, $\text{cur\_position}$ will be updated to the termination address of $\alpha_1$, i.e., the starting address of $\alpha_2$, and thus the behavior rule related to $\alpha_2$ will be triggered and $\alpha_2$ will start to execute subsequently.

2. For a composite activity, such as <sequence>, <switch> and <while>, the behavior rules related to the activity can assure that all offspring activities involved in the composite activity will be performed in the ways specified by the composite activity’s specification. Without losing generality, suppose the composite activity is a <while> activity. According to the behavior rules related to a <while> activity, when the condition for iteration of the <while> activity is satisfied, the first offspring activity occurring in the <while> activity will be activated to start and after the last offspring activity finishes, the behavior rules will re-assign the starting address of the
<while> activity to cur_position. Otherwise, i.e., the condition of the <while> activity is not satisfied, cur_position will be assigned with the termination address, and thus the control of the execution of the process will leave the <while> block and continue to the next activity. Therefore, the behavior rules corresponding to the <while> activity will control the activity’s execution as the process specification has specified.

3. For a composite activity of <flow>, the offspring activities involved in the <flow> activity will be performed concurrently based on the semantics of the <flow> activity. While generating the behavior rules related to a <flow> activity, let all of the offspring activities of the <flow> activity have the same starting address and then all of the behavior rules related to the offspring activities have the same triggering condition as well. Thus, all of those behavior rules will be triggered simultaneously, which will lead to the concurrent executions of all of the offspring activities of the <flow> activity.

3.3. Adaptation Units

As mentioned in previous sections, a BPEL4WS process is predefined and does not evolve. Since a software system built from web services is situated in the dynamic and open Internet environment, it is a difficult problem to ensure the functional availability of the system with a single and fixed web service process. Therefore, the adaptable composition framework allows the process definer to use a set of adaptation units specifying new adaptation strategies to equip an existing web service process at runtime. The concept of adaptation unit is similar to an application’s plug-in, which can be loaded or removed at runtime, so not only the designers, but also the deployers and maintainers can modify the alternatives specified in the units or apply new adaptation strategies to the business process.

In the implementation, several adaptation units have been developed to facilitate the adaptations of business processes. Due to space limitation, just two typical units, named as “ProcessSwapping” and “ServiceSwapping”, are introduced in the following context.

**ProcessSwapping (PS):** To achieve a certain goal, people may define multiple candidate processes. When
facing several processes, the process definer can specify specific selection strategies. For instance, a strategy can be specified based on some QoS related criteria so that a candidate process will be selected according to the criteria. Simply, candidate processes can be selected in the literal order whenever the preceding process is failed.

**ServiceSwapping (SS):** Similarly, a group of replaceable services and a corresponding selection policy for a certain service can be specified in the process when the service needs to be updated or replaced online. When a candidate service is selected to replace the existing service, it may introduce mismatching interactions and those mismatch problems must be solved.

By using these strategy units, the behaviors of processes can be adjusted and the system goals can be achieved in different situations and with different qualities of services without interrupting the run of processes.

For the sake of understanding, an example is presented to illustrate how to use adaptation units. Figure 2 shows a simple process “MusicStore” providing music purchase service. First it receives the necessary information from the customer, such as the credit card number and a list of song names. Next it fetches the MP3 format music files from a music library. Then it invokes a service to convert these MP3s into WAV format files. Lastly it uses a CD-burning service to burn the WAVs onto a CD.

![Figure 2. The Process for the Online Music Store](image)

### 3.3.1. Swapping Activities with Sub-Processes

A certain (sub-) goal may be fulfilled in different ways using different service processes. For example, in the above MusicStore process, when the format conversion service (e.g., MP3toWAV) is unavailable or the customer requests to obtain higher quality WAV music, another sub-process can be defined as the alternative solution for the format conversion, for instance, converting MP3 into WMV format first and converting WMV
into WAV next (see Figure 3).

![Diagram](image)

**Figure 3. Extending the MusicStore using Adaptation Units**

The `<ProcessSwapping>` template is used for describing such an activity consisting of several candidate processes, which also employs the BPEL4WS for the descriptions, and then an adaptation unit for process swapping can be specified via concretizing the template. For example, in List 1, the adaptation unit encapsulates both the original web service “MP3toWAV” and the alternative sub-process (“MP3toWMV”→”WMVtoWAV”). Property `<target>` specifies that this unit will be applied to the “mp3toWav” activity in the MusicStore business process. The property `<SelectionPolicy>` defines the selection policy among the different sub-processes. How to define a specific selection policy is described in the following sub-section. Without a policy designated, candidate processes will be selected in the literal order by default and will be skipped when unavailable. Thus, before the ProcessSwapping adaptation unit is loaded, the process will only try to invoke the service “MP3toWAV”; hereafter the adaptation unit is loaded, the process will try to invoke “MP3toWAV” or execute the sub-process alternatively according to the specified selection policy.

```
<ProcessSwapping name = “PS_MusicConversion” >
   <target> “mp3toWav” </target>
   <SelectionPolicy> … </SelectionPolicy>
   <candidateProcess name = “WF1” >
      …… <!-- BPEL4WS fragment, invoking a service for converting music from mp3 to wav.-->
   </candidateProcess>
   <candidateProcess name = “WF2” >
      …… <!-- depicting a sub-process converting mp3 to wmv, and then wav. -->
   </candidateProcess>
</ProcessSwapping>
```

**List 1. ProcessSwapping Template**
3.3.2. Swapping Activities with Individual Activities

In a ProcessSwapping adaptation unit, the process is allowed to select different sub-process to achieve the same goal. However, in many cases, when a web service becomes unavailable during the execution of the process due to unstable network disconnections or service refusals, it is more efficient and acceptable just to substitute the unavailable service with another service. To achieve this, the <ServiceSwapping> template is used to specify backup services and an optional selection policy for a service. In the above example (Figure 3), it can provide several CD-burning services to increase the flexibility and reliability of the whole process. List 2 gives a description of the corresponding adaptation unit consisting of several candidate services.

```
<ServiceSwapping name = "SS_CDBurning" >
  <target> "Burn CD" </target>
  <SelectionPolicy > … </SelectionPolicy >
  <candidateService > <invoke name = "BurnCDServ2" …/> </candidateService >
  <candidateService > <invoke name = "BurnCDServ3" …/> </candidateService >
</ServiceSwapping >
```

List 2. ServiceSwapping Template

Generally, in the process specification, it is assumed that web services involved in the process interact seamlessly, i.e., no mismatching interactions are involved. However, once a certain pre-assumed web service becomes unavailable, and another candidate service is invoked for interaction with other web services in the process, this interaction might be impossible if the interaction pattern (e.g., message names, parameters, and etc.) of the candidate service is different from its replaced one.

To ensure the achievements of system goals, it should assure that candidate services could accurately accept the input messages and return the expected output as the replaced service. For those services already in existence, they should interact with the candidate service as if they were interoperating with the originally specified one. As the service replacement is dynamically carried out during the process execution, it is necessary to eliminate mismatching interaction automatically so that the replacement could be performed transparently and the candidate services could be integrated and run in the process correctly and seamlessly.
However, because there exists great difference between various web services’ message formats, it is difficult to provide a fully automatic mechanism for the transformation between these messages. Moreover, when more input parameters are required for the candidate services, it is impossible to supply the additional information automatically.

For example, in the previous MusicStore process, the CD-burning service BurnCDServ1 accepts messages containing four parameters: “music” specifies the URL list of to-be-burned music files, “delivery” specifies the delivery choice, such as ordinary shipping or express mailing, “address” provides the delivery destination, and “payment” indicates how to pay for the purchase. Assume that a backup service BurnCDServ2 is specified for BurnCDServ1. Differently, BurnCDServ2 requires three parameters differently. In BurnCDServ2, the destination address is also stored in parameter “address”, but the music list is stored in a different parameter “songs”. This service is free but it can be used only if a valid “user” is authenticated. Nevertheless, an anonymous account “guest” can be used to access the service. In addition, the service confines the delivery type to be ordinary shipping so it does not need to specify the delivery type.

In the above example, to invoke the candidate service correctly, the music list and address information need to extract from the former message and to fill the extra-required account information. Currently, there is some research work on automatic discovery, selection and interaction adaptation based on OWL-S and domain ontology (e.g., [16]). However, at the current stage, one cannot expect that all web services have had explicit semantic descriptions. Besides, with current semantic model, it is difficult to cope with such a situation when some extra parameters are required in the candidate service (e.g., the user account in BurnCDServ2).

For automatically converting different messages and making the service replacements as transparent as possible to users, a compromised solution is employed. This solution is based on the observed fact that users often rely on several relatively fixed service providers for the service rather than dynamically discover and use unfamiliar services. Therefore, the solution allows user to specify replacement services and designate how to
convert messages in the adaptation template. When some extra parameters are required, users are allowed to provide default values for the extra parameters.

The message conversion between the original service and its backup service is specified as name mapping in the SwappingService unit. A mapping is designated in an optional <msgMapping> markup in <candidateService>, as shown in List 3. The property <inputMessage> depicts how to make message conversion, and <part> in <inputMessage> describes how to assign the value of one parameter (source) in the original service’s message to another parameter (destination) in the backup service’s message. If there is no such correspondent in the original message, it need designate a default value for the parameter in the backup service’s message, such as user id and password in BurnCDServ2. Inversely, <outputMessage> depicts how to convert results returned from the backup service to be acceptable for the original service.

```
<SwappingService name=“SS_CDBurning”>
  <target>“Burn CD”</target>
  <SelectionPolicy>…</SelectionPolicy>
  <OriginalService>
    <invoke name=“BurnCDServ1”
      portType=“ws1:BurnCDService”, operation=“burn_cd”
      inputVariable=“OrderToServ1”, outputVariable=“Serv1Result”/>
  </OriginalService>
  <candidateService>
    <invoke name=“BurnCDServ2”
      portType=“ws2:BurnCDService”, operation=“burnCD”
      inputVariable=“OrderToServ2”, outputVariable=“Serv2Result”/>
    <msgMapping>
      <inputMessage>
        <part source=“address”, destination=“address”>
        <part source=“music”, destination=“songs”>
        <part source=“***”, destination=“user” default=“guest”>
        <part source=“***”, destination=“password” default=“”>
      </inputMessage>
    </msgMapping>
    <outputMessage>…”</outputMessage>
  </candidateService>
</SwappingService>
```

List 3. Eliminating Interaction Mismatches between Services
3.3.3. Selection Policy

In the adaptation units described above, they often designate some selection policies for selecting different alternative sub-processes or individual activities when originally specified services are unavailable or the process is required to adapt to new situations, for instance, to provide higher qualities of services for users.

In the adaptation units, the markup `<SelectionPolicy>` is used to specify those selection policies. A `<SelectionPolicy>` gives a rule set and every rule is represented by a `<rule>` markup. For a selection rule, its triggering condition may be about the state of the runtime environment (e.g., the network status, the service workload, and so on) or concerned with the business strategies.

For example, in the above process, the PS_MusicConversion unit may be configured with multiple candidate sub-processes providing different quality of service, and it is up to the process logic to decide which one to be invoked according to the service requester's identity. When the user is a VIP customer (e.g., “customerRank > 1”), the MusicStore will call a candidate with a higher quality (see List 4).

The content in `<result>` points out the candidate (a certain candidate process or service) to be selected when the condition is satisfied.

```
<SelectionPolicy>
  <rule name = “forVIP”>
    <condition>
      bpws:getVariableProperty(request, customerRank) > 1
    </condition>
    <result>
      <select name = “WF1”> //Assuming WF1 provides a high quality service
    </result>
  </rule>
</SelectionPolicy>
```

List 4. Selection Policy

3.3.4. Translating Adaptation Units into Behavior Rules

Since the behavior of agents is controlled via behavior rules, the specified adaptation units should be transformed into rules so that agents could infer and trigger their behaviors and then realize the adaptations of
the processes according to the adaptation units. Therefore, a new category of behavior rules is added into an agent’s rule set, i.e., adaptation rules, and meanwhile an algorithm is implemented to generate adaptation rules from the adaptation units automatically.

To enable the process to be adaptable according to the adaptation units, a mechanism should be offered for modifying the behavior rules of agents. An adaptation rule can be specified as follows.

<table>
<thead>
<tr>
<th>Condition</th>
<th>The condition that an adaptation requirement appears.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Make adaptations to the execution of the process.</td>
</tr>
</tbody>
</table>

In the action part, agents may delete some behavior rules related to the original services and add some new behavior rules related to the candidate sub-processes or individual services.

### Algorithm 3: Translating an Adaptation Unit into Rules.

1. For a <ProcessSwapping> unit, assume that the <ProcessSwapping> activity’s id (or name) is \( P \), there exists \( N \) candidate sub-processes, \( CP_1, CP_2, \ldots, CP_n \), and \( C_i \) \((i = 1 \ldots N)\) is the condition for selecting the candidate sub-process \( CP_i \).
   a. If the user defines a selection policy in the unit, then an adaptation rule can be obtained.
      
      cur_process=\( P \)id and cur_position = p.start and \( C_i \) \( \Rightarrow \) Select \( CP_i \) to execute.
   b. Otherwise
      i. The first candidate sub-process will be selected first.
         
         cur_process=\( P \)id and cur_position = p.start and \( CP_1 \) has been selected. \( \Rightarrow \)
         Select \( CP_1 \) to execute.
      ii. And the succeeding sub-process will be selected if its preceding sub-processes all have failed.
          
          cur_process=\( P \)id and Failed(\( CP_i \)) \( \Rightarrow \) Select \( CP_i \) to execute.
      iii. A sub-process fails if any of its involving activity fails.
           
           cur_process=\( P \)id and Failed(Activity) \( \Rightarrow \) Failed(Activity, parent)
   c. When a sub-process (\( CP_i \)) is selected after another sub-process (\( CP_j \)), generate the following adaptation rules to modify the rule set.
      
      cur_process=\( P \)id and Selected(\( CP_i \)) and De-selected(\( CP_j \)) \( \Rightarrow \)
      Delete all behavior rules related to \( CP_j \);
      Use algorithm 1 to generate new behavior rules related to \( CP_i \);
      Push cur_process and cur_position into the stack;
      cur_process = \( CP_i \)id; cur_position = 1.
      cur_process = \( CP_i \)id and current_position = -1 and the stack is not empty \( \Rightarrow \)
      Pop the parent process’s id and current position out of the stack;
      Assign them to cur_process and cur_position, respectively.

      The second rule will be triggered when the selected process returns successfully.

2. For a <ServiceSwapping> unit, the adaptation rules can be generated similarly as above.
4. Implementation of the Agent-based Adaptable Process Execution Framework

4.1. Lightweight Agent’s Running Support

As shown in Figure 1, for an agent controlling and managing the execution of a business process, there are mainly five functional parts, including the interpreter, the sensor, the rule engine, the behavior manager and the effector. The interpreter transforms the input specification into agent’s rules. The sensor is responsible for perceiving the states of the environment, capturing the results of the invocations of web services, intercepting communication messages between web services, and probably resolving mismatching messages when some adaptation units are plugged into the process. Then the aforementioned data are transformed into a format which can be recognized by the rule engine and stored in its knowledge base. The rule engine is responsible for matching and activating the rules to implement the business process, and the behavior manager schedules the behaviors agent should execute. Finally, the effector takes charge of invoking web services according to the behavior manager’s agenda.

Ideally, an agent can be used to support any business processes and there is no need to implement a specific agent for each process on one hand, and an agent supporting a process can be realized without coding on the other hand, i.e., people need only specify the process and the ways to invoke web services and probably some adaptation units, and then the corresponding agent can be generated automatically based on the specifications.

Therefore, a platform is implemented on a J2EE-compliant application server (named PKUAS) to support lightweight agents, in which agents are not solid entities and they will be substantiated when the specifications of agents are loaded.

The PKUAS is a component-based reflective middleware platform and provides an environment for deploying and executing interoperable components [14]. In the PKUAS, containers provide runtime spaces for instances of components and meanwhile containers have the ability to capture all details concerning components via the reflection mechanism. In the implementation, containers load the specifications of agents to generate
agent instances and support the runs of agents.

Lightweight agents for building adaptable business processes are implemented on the PKUAS as follows (Figure 4).

In the running support, the agent’s specification includes the specifications of the process, the invocation specifications of web services, and the optional adaptation units.

The agent container loads the agent’s specification to generate the behavior rules of the agent. The interceptors in the agent container implement both the sensor and the effecter. The interceptors capture related information from the environment and then transfer it to the rule engine for triggering the behavior rules related to the process. In addition, the interceptors are also responsible for eliminating mismatching message communications.

Because every agent supporting the adaptable business process needs a rule engine to manage its behavior rules, the rule engine and the rule base as well as the behavior base are implemented as public services. The rule base stores the behavior rules of the agent generated by the container and the behavior base saves the information about how to invoke individual web services. The rule engine controls the executions of the behavior rules stored in the rule base, and activates the invocations of web services by the ways specified in the
behavior base. In the implementation, it integrates a rule engine for Java platform, Drools [9], which can be used to reason about the agent’s behaviors based on declarative rules.

As mentioned above, the agent’s behavior base just specifies the ways to invoke web service. To support adaptable business processes, the invocations of web services are also realized on the same middleware platform. Therefore, the Apache AXIS [2] and WSIF [31] are integrated: the former is as the SOAP engine while the latter allows clients to invoke services focusing on the abstract service description with its Dynamic Invocation Interface (DII). More details about the web service infrastructure supported on the PKUAS can be found in [13].

In the running support, a public service for transaction processing is also implemented. As mentioned above, a web service will be replaced if it is unavailable, its invocation has failed, or there is some adaptation policy specified. However, when the replacement takes place before a web service finishes, the incomplete execution of the web service may put some side effects on the execution of the whole business process. The transaction processing public service is responsible for the atomicity of the execution of a web service and it can ensure that an incompletely executed web service can be replaced safely.

In principle, when a web service is unavailable or its invocation is failed, the agent need only trigger the adaptation rules to activate new web service(s) to replace the failed one. However, when a web service is being invoked and the user submits some new adaptation policies related to the web service, replacing the being invoked web service will do harm on the execution of the business process. To enable the agents load user-defined adaptation units and replace web services at runtime, the running support platform even provides an online evolution mechanism [23][29]. Thus, the transaction processing service and the online evolution mechanism will together guarantee both the safe dynamic replacements of web services and the successful execution of the business process.

4.2. Execution of Adaptable Processes

In the agent based framework, the execution of an adaptable process covers 1) starting the process, 2)
controlling the execution of the process, e.g., transiting from one activity to another activity and adapting the process via replacing candidate sub-processes or services, 3) performing atomic activities, and finally 4) terminating the process.

1). To initiate a business process, the agent will set its internal variables cur_process to be the process’s id and cur_position to be 1. Thus the rule relevant to executing the first activity of process execution will be triggered.

2). When the process is activated, the rule engine of the agent will be responsible for the execution of the process via inferring and triggering the behavior rules of the agent.

3). The behaviors of the process are exhibited as a sequence of executions of a group of atomic activities (i.e. invoke, receive and so on), which will be fulfilled by the effector of the agent.

4). When cur_position = -1, all the rules would no longer be triggered. If the process carry out smoothly, agent would ultimately enter the status that cur_position = -1.

4.3. The Example

Through the algorithms described in the previous sections, the above music purchasing process without plugging adaptation units can be transformed into rule sets as shown in List 5.

List 5. Rules Translated from the BPEL4WS Specification

After the process-swapping unit “PS_MusicConversion” is loaded, the agent will generate the following adaptation rules and replace the above 4th rule with them.
c. cur_process = “WF1” & cur_position = -1 → Pop(cur_process, cur_position);
   cur_position = 5;

4.2. Failed (WF1) → deleteRules(4.1#, a#, b# and c#); Select (WF2);
  d. Selected (WF2) → Push(cur_process, cur_position);
     cur_process = “WF2”; cur_position = 1;
  e. cur_process = “WF2” & cur_position = 1 → do (MP3toWMV); cur_position = 2;
  f. cur_process = “WF2” & cur_position = 2 → do (WMVtoMAV); cur_position = -1;
  g. cur_process = “WF2” & cur_position = -1 → Pop(cur_process, cur_position);
     cur_position = 5;

List 6. Adaptation Rules related to the Process Swapping Unit

And after the service-swapping unit “SS_CDBruning” is loaded, the agent will add the following behavior rules into the rule set.

5.1. Failed (Burncd1) → Do (Burncd2); cur_position = -1;
5.2. Failed (Burncd2) → Failed (“MusicStore”)

List 7. Adaptation Rules related to the Service Swapping Unit

From these rules related to the adaptation rules, it can be found that, by dynamically removing behavior rules related to failed web services and adding rules for adapting to new situations, the agent supports a more adaptable business process for the online music store.

5. Conclusions

In this paper, an agent-based framework for composing web services to support adaptable business processes was described. In the framework and implementation, the adaptations of the business process are specified in independent adaptation units and the agent can load and interpret those adaptation units at runtime. Thus, the executions of business processes can be adapted dynamically.

Since changes of a process are possible on both the process level and the service level, two adaptation templates were designed and implemented to enable substituting individual services with some candidate services or processes at runtime, where the substitutions and selections of candidates are guided by using specific business policies. Based on the pluggable architecture, new templates can be easily developed to extend the framework with new adaptation strategies. The work is going on to define more adaptation templates, for
example, to allow a composite activity or a group of individual activities as a whole to be replaced with some candidate sub-processes.

Because the framework supports changing the business logic at runtime, runtime verification of the modification is required to check whether these changes may cause negative effects on the process’s execution, for instance, leading to deadlocks. In the current approach, an adaptation unit defines a fine-grained granularity to limit process change in a controllable range. Take the ServiceSwapping unit as example: if the unit’s internal state is safe and the candidate services are function-equivalent, then the modified process will has the same properties as the original one after applying this unit. Many researches have focused on web service process verification [5][12], but there is no existing approach to checking process properties at runtime. The future work on this issue is to give a formal basis for modeling the semantics of changes and support online verification.

In the current implementation, the adaptation units are manually defined by the user, which demands the user should be familiar with the BPEL4WS specification language. Therefore, a GUI is being developed for the user to draw the business process with adaptation units. The GUI will automatically generate the specification of the adaptation units in the BPEL4WS.

As the transformation algorithms presented in this paper are responsible for translating the process specification into agent behavior rules, in the framework, the runtime performance difference between the approaches and non-agent approaches can be considered to some extent as comparing a declarative rule-driven approach with a direct-call approach. While the declarative rule-driven approach brings the benefit of flexibility and maintainability, it has some additional cost in rule interpreting and matching. Fortunately, modern rule engines are fast enough by using the “Rete” algorithm [11]. So the framework’s performance will be in an acceptable level.

In the ongoing research, more practical application case will be studied to illustrate the efficiency and applicability of the approach. The quantitative evaluation of the framework will be done in parallel.
Acknowledgements

This work is partially sponsored by the National Basic Research Program of China (973) under Grant No. 2002CB312003, and the National Natural Science Foundation of China (Grant No. 60125206, 60233010, 60303004, 60403030, and 90412011).

References


[21]. OWL Services Coalition, *OWL-S: Semantic Markup for Web Services The OWL Services Coalition* See http://www.daml.org/services/owl-s/1.0/, 2003


[28]. W3C: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core language*, W3C Working


Hong Mei received a Bachelor’s degree and a Master’s degree in Computer Science from Nanjing University of Aeronautics and Astronautics (NUAA) in 1984 and 1987, respectively, and a Doctorate degree in Computer Science from Shanghai Jiao Tong University in 1992. From 1992, he joined in Peking University (PKU) and now he is full professor at the Department of Computer Science and Technology and dean of the School of Electronics Engineering and Computer Science in PKU. His current research interests include: Software Engineering and Software Engineering Environment, Software Reuse and Software Component Technology, Distributed Object Technology, Software Production Technology, and Programming Language.

Xi Sun is a Ph.D. candidate and his research interests include software engineering, software component technology, and intelligent software systems.

Xuanzhe Liu is a Ph.D. candidate and his research interests are in service-oriented architecture (SOA), Web
services, and feature interactions in middleware-based systems.

**Wenpin Jiao** got his BE and ME in Computer Software at East China University of Science and Technology in 1991 and 1997, respectively, and his Ph.D. in Computer Science at Chinese Academy of Sciences in 2000. From 2000 to 2002, he worked with the University of Victoria as Post-Doctoral Fellow. Now he is associate professor working with the School of Electronics Engineering and Computer Science of Peking University. His research interests mainly include software engineering, multi-agent systems, automatic computing, and formal methods.

**Gang Huang** received a B.S. and an M.S. in Computer Science from the Northwestern Polytechnical University in 1997 and 2000, respectively, and his Ph.D. in 2003 from the School of Electronics Engineering and Computer Science of Peking University. Now, he is Associate Professor in the School of Electronics Engineering and Computer Science of Peking University. His research interests are in the area of Distributed computing with emphasis on middleware, including the construction and management of middleware, and software engineering with emphasis on component based development and software architecture. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and China Computer Federation (CCF).