Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Electronic Commerce Research and Applications 3 (2004) 214–231
Electronic Commerce Research and Applications
www.elsevier.com/locate/ecra

A three-level specification approach for an environment of software agents and Web services

Zakaria Maamar [a,*], Quan Z. Sheng [b], Boualem Benatallah [b], Ghazi Al-Khatib [c]

[a] *Software Agents Research Group, College of Information Systems, Zayed University, Dubai, United Arab Emirates*
[b] *School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia*
[c] *Qatar College of Technology, Doha, Qatar*

Received 22 April 2003; received in revised form 19 November 2003; accepted 3 December 2003
Available online 31 December 2003

Abstract

This paper presents an approach for the specification of a software agent-based and Web service-oriented environment. A software agent is an autonomous entity that acts on user's behalf. Whereas a Web service is an accessible application that other applications and humans can discover and trigger. Users in collaboration with their agents compose Web services into high-level business processes denoted by composite services. The participation of Web services in a composite service is based on several selection criteria such as the execution cost of a Web service and the location of the resources on which a Web service will be performed. Prior to that selection, the specification approach puts forwards three levels: intrinsic, organizational/functional, and behavior. Besides the specification approach, the composition of Web services is illustrated in this paper with service chart diagrams.

Keywords: Web services; Software agents; Specification; Composition; Location

1. Introduction

Our long-term research objective is the deployment of anytime, anywhere applications through the design and development of environments for stationary and mobile users. To reach this objective, we consider two major components to constitute such environments: *software agents* and *services*. We decompose software agents into two types [12]: stationary and mobile. Further, we decompose services into two types: *Web services* [21] and *Mobile services* (M-services) [15].

With the rapid development of information technologies, several businesses are deploying Web-based applications for more automation, efficient business processes, and global visibility. Web services are among the technologies that help businesses in being more Web-oriented. A Web service is an accessible application that other applications and humans as well can discover and trigger [8].

* Corresponding author. Tel.: +971-42082461; fax: +971-42640854.
E-mail addresses: zakaria.maamar@zu.ac.ae (Z. Maamar), qsheng@cse.unsw.edu.au (Q.Z. Sheng), boualem@cse.unsw.edu.au (B. Benatallah), alkhatib@qu.edu.qa (G. Al-Khatib).

1567-4223/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.elerap.2003.12.002

Various technologies are behind the success of Web services including Web Services Description Language (WSDL), Universal Description, Discovery and Integration (UDDI), and Simple Object Access Protocol (SOAP) [9]. These technologies support the definition of Web services, their advertisement, and their binding for triggering purposes. Web services have the capacity to be composed into high-level business processes usually known as *composite services*. For example, a vacation scenario calls for the collaboration of at least four Web services: flight reservation, hotel booking, attraction search, and user notification. These Web services need to be connected according to a certain flow of control. Flight reservation is first completed, before hotel booking and attraction search can be both initiated.

With the latest development in mobile and wireless technologies, a new generation of Web services are put forward on the market for the benefit of persons who are usually on the move (e.g., sales representatives). This kind of persons rely on mobile devices such as cell-phones to conduct their day-to-day operations. M-services denote this type of Web services and are meant to be either: (i) triggered remotely from mobile devices for execution purposes, or (ii) wirelessly transferred from provider sites to the mobile devices of users on which their execution is carried out [15]. Anytime, anywhere applications should support users in satisfying their needs regardless of their location and the type of devices (fixed or mobile) they use.

It is observed in [20,23] that composing multiple services rather than accessing a single service is essential. Searching for the relevant services, integrating them into a composite service, triggering them, and monitoring their execution are among the operations that users will be responsible. Most of these operations are complex, although repetitive with a large segment suitable to computer assistance. Therefore, software agents are deemed appropriate candidates to assist users in their operations. For the needs of agentifying services, we put forward two types of software agents; those that act on behalf of users are called *user-agents* and those that act on behalf of providers of services are called *provider-agents*. Throughout this paper, we argue that the integration of software agents and services into the same environment requires a specification approach that provides assistance to designers. The specification is undertaken at three levels: *intrinsic*, *organizational/functional*, and *behavior*. Each level has a set of properties that vary according to the component (i.e., agent or service) to which the level is applied.

- Agent: the intrinsic level has the properties that identify an agent as an independent entity. The organizational level has the properties that identify an agent as an element of a community of agents. Finally, the behavior level has the properties that describe the reactions of an agent to the interactions in which it participates.
- Service: the intrinsic level has the properties that identify a service as an independent entity. The functional level has the properties that describe a service as a member of a composite service. Finally, the behavior level has the properties that describe the states of a service when it is the subject of interactions between agents.

In this paper, the following elements are discussed. What are the appropriate mechanisms for specifying an environment of Web services? What is the value-added of software agents to these mechanisms? What are the selection criteria to integrate component services into composite services? How to identify the resources on which the component services will be performed? And finally, how to validate the specification approach using prototyping techniques? The remainder of this paper is structured as follows. Section 2 overviews some core concepts such as software agents and service composition approaches. Section 3 presents the three-level specification approach. Section 4 discusses the process of agentifying an environment of Web services. Work in progress and related work are respectively outlined in Sections 5 and 6. Finally, Section 7 summarizes our contributions and draws our conclusions.

2. Background

2.1. Software agents

A software agent is a piece of software that autonomously acts to perform tasks on user's behalf

[12]. Many software agents design is based on the approach that the user only needs to specify a high-level goal instead of issuing explicit instructions, leaving the how and when decisions to the agent. A software agent has various features that make it different from other traditional components such as autonomy, goal-oriented, collaborative, flexible, and mobile.

2.2. Services

Regardless of its type (i.e., Web or mobile), a service consists of operations to perform according to a certain input and a certain chronology. Potential users have to know how to request a service for execution, but neither how to operate the service nor how the service operates or is operated. In this paper, *C-service* denotes a composite service, whereas *P-service* denotes a primitive service that is either a Web service or an M-service.

2.2.1. Web services

A Web service in [3] is an accessible application that other applications and humans can discover and trigger. Benatallah et al. associate three properties with a Web service: (i) independent as much as possible from specific platforms and computing paradigms; (ii) developed particularly for inter-organizational situations; and (iii) easily composable, i.e., its composition with other Web services does not require the development of complex adapters.

2.2.2. M-services

Maamar and Mansour provide in [15] two definitions for an M-service. The weak definition is to remotely trigger a Web service from a mobile device for execution. In that case, the Web service acts as an M-service. The strong definition is to transfer a Web service from its hosting site to a mobile device where its processing takes place. In that case, the Web service acts as an M-service that is: (i) transportable through wireless networks; (ii) composable with other M-services; (iii) adaptable according to the computing features of mobile devices; and (iv) runnable on mobile devices. In this paper, we only consider the M-services that comply with the weak definition.

Fig. 1 shows snapshots of a mobile service running on a cell-phone. The service provides information to tourists visiting a city for instance Dubai. Upon request of tourists, the service is downloaded into their mobile devices.

2.2.3. C-service vs. P-service

A C-service aggregates multiple component P-services. Since the M-services that we refer to in this paper comply with the weak definition, the P-services correspond only to Web services. Instead, the C-services are made available to users for triggering purposes in two different versions: Web version for stationary users and M-version for mobile users. As part of our work of specifying the composition of services, we developed *service chart diagrams* [14]. Details on these diagrams are given below.

2.2.4. Service chart diagram

A service is expressed using a service chart diagram, which leverages the state chart diagram [1] of UML [11]. In a service chart diagram, the emphasize is on the context surrounding the execution of a service rather than only on the states that a service takes (Fig. 2).

A service chart diagram wraps the states of a service into four perspectives, besides the *state* perspective that is in fact the state chart diagram of the service. The *flow* perspective corresponds to the execution chronology of a composite service (previous services/next services attributes – M/O respectively stands for Mandatory and Optional). The *business* perspective identifies the organizations that are ready to provide a service (business attribute). The *information* perspective identifies the data that are exchanged between services (data from previous services/data for next services attributes). Because of the type of services (i.e., mandatory and optional), the information perspective is tightly coupled to the flow perspective with re-

[1] A state chart diagram is a graphical representation of a state machine that visualizes how and under what circumstances a modelled element (e.g., a class, a system, or a business process) changes its states. Furthermore, a state chart diagram is used for showing which actions are executed as a result of event occurrence.

Fig. 1. Snapshots of tourist mobile-book.

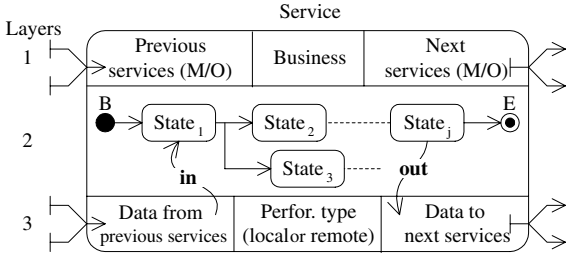

Fig. 2. Service chart diagram.

gard to mandatory data vs. optional data. Finally, the *performance* perspective illustrates the ways a service is invoked for execution whether remotely or locally (performance type attribute, Fig. 3, [13] for more details on a service's invocation types).

2.3. Approaches of service composition

A composite service consists of component services that are either primitive or composite. We outline below the approaches of service composition [5].

2.3.1. Proactive composition vs. reactive composition

Proactive composition is an off-line process that gathers in advance the available component

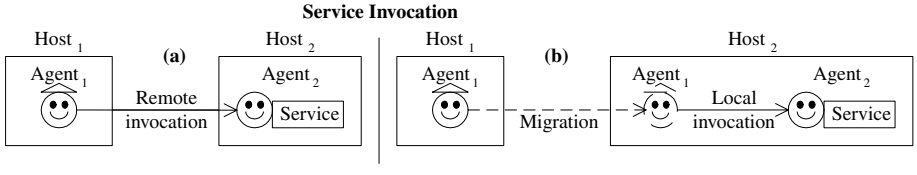

Fig. 3. Service invocation types: remote vs. local.

services to constitute a composite service. This means that the composite services are pre-compiled and ready to be executed upon users' requests. In a proactive composition, the component services are usually stable and may even be running on resource-rich platforms. With regard to reactive composition, it is the process of creating composite services on-the-fly. A composite service is devised on a request-basis from users. Because of the on-the-fly property, a component manager that ensures the identification and coordination of the component services is required. Despite a "certain" complexity of reactive composition, it presents several advantages such as the awareness of the current status of the component services and the possibility of optimizing certain runtime arguments such as bandwidth use and data transfer routes.

2.3.2. Mandatory composite service vs. optional composite service

A mandatory composite service illustrates the compulsory participation of all the component services to the execution process. Because it is expected that the component services will be spread across the network, the reliability of the execution process of each component service affects the reliability of the whole composite service. On the other hand, an optional composite service does not necessarily require the participation of all the component services. Certain component services can be skipped during the execution for various reasons such as possibility of replacement or non-availability.

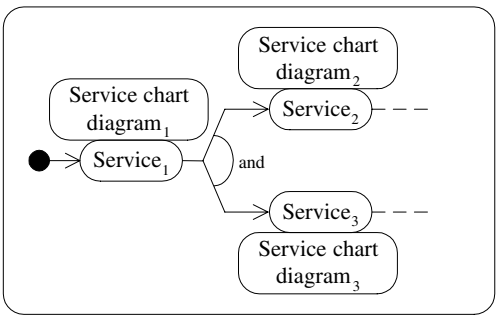

Fig. 4. State chart diagram of a composite service.

We recall that a composite service consists of several component services. Therefore, the process model underlying a composite service is specified as a state chart diagram whose: (i) states are associated with the service chart diagrams of the component services, and (ii) transitions are labelled with events, conditions, and variable assignment operations. Fig. 4 illustrates the state chart diagram of a composite service that integrates the service chart diagrams of its component services.

3. The three levels of the specification approach

3.1. Motivation

To satisfy users' and businesses' needs, multiple advanced components are put forward to devise new types of applications. Moreover, the diffusion of mobile telecommunications and mobile access to the Web has widened the heterogeneity of client devices. Devices span from traditional workstations and PCs, to laptops, personal digital assistants and smart phones, with wired/wireless continuous/intermittent connectivity.

With the latest progress in information technologies, needs of users are getting diverse and complex. To satisfy these needs, advanced components are required for developing a new type of applications. In our work, we decided using software agents and services. Agents are autonomous entities that take initiatives in solving problems. Whereas services are computing packages that are intended to be composed and executed. The composition of services into high-level business processes is complex. Indeed, this requires searching for the relevant services, integrating them into a composite service, triggering the services at the right time and in the right order, and monitoring the execution progress of the composite service for exception handling. All these operations can be outsourced to software agents. They have the capacities to handle them such as autonomy, mobility, and collaboration. Putting services and agents in the same environment indicates the importance of having a specification approach that we apply in two steps. In the first step, the

specification approach is applied to agents and services taken independently from each other. Whereas in the second step, the specification approach is applied to agents and services taken in a combined way.

3.2. Levels of the approach

The three levels of the specification approach are as follows: intrinsic, organizational/functional, and behavior. In what follows, the properties of each level are described and afterwards illustrated with examples.

3.2.1. Software agent

Table 1 summarizes the properties of the specification approach when applied to software agents.

- The intrinsic level consists of three properties: identifier (identifies an agent with regard to a certain Internet domain), role (lists the responsibilities of an agent), and type (indicates if an agent is of type provider or user).
- The organizational level consists of two properties: visited domain, and not-visited domain. When a service is locally triggered, this means that the requesting agent is visiting the domain of the requested agent of the service. The information on that domain updates visited-domain property. The opposite happens if a service is remotely requested; not-visited-domain property is updated.
- The behavior level consists of one property: state chart diagram (lists the states that an agent takes when it interacts with its peers).

3.2.2. Service

Table 2 summarizes the properties of the specification approach when applied to services.

Table 1
Properties of the specification approach applied to software agents

Level	Properties
Intrinsic	Identifier, role, type
Organizational	Visited domain, not-visited domain
Behavior	State chart diagram

Table 2
Properties of the specification approach applied to services

Level	Properties
Intrinsic	Identifier, description, type, input arguments, output arguments, cost
Functional	Component link, mandatory causal link, optional causal link
Behavior	State chart diagram

- The intrinsic level consists of seven properties: identifier (identifies a service with regard to all the services offered to users), description (provides an overview of a service), type (indicates if a service is of type composite or primitive), input arguments (lists the number and type of arguments that are submitted when a service is triggered), output arguments (lists the number and type of arguments that a service returns after processing), and cost (corresponds to the charges of executing a service).
- The functional level consists of the three properties: component link (only applies to composite services and identifies the component primitive-services), mandatory causal link (identifies the primitive services that are attached to a primitive service; these primitive services *have* to be added to the composite service in case the primitive service participates in this composite service), and optional causal link (identifies the primitive services that are attached to a primitive service; these primitive services *can* be added to the composite service in case the component service participates in this composite service. [2]
- The behavior level consists of one property: *state chart diagram* (describes the states that a service takes when the service is the subject of conversation between agents).

3.2.3. Connection between agents and services

The connection occurs at the three levels namely intrinsic, organizational/functional, and behavior.

[2] A person attending a conference in city X could be interested in visiting the famous places of that city even if the person did not explicitly mention his interest.

- *Intrinsic level*: An agent of type provider has to know the services it offers. Thus, *service* property, which refers to these services, is added to the intrinsic level of the agent.
- *Organizational–functional level*: To satisfy his needs, a user triggers a composite service to be assigned to an agent of type user. This agent knows the primitive services of the composite service using component link, mandatory causal link, and optional causal link properties of the functional level of the service. The agent has to select the primitive services needed according to execution cost property. In addition, the agent has two options to trigger a primitive service: (i) locally within the same domain of the primitive service, or (ii) remotely from a different domain of the primitive service. The type of invocation enables updating visited domain and not-visited domain properties of the organizational level of the agent.
- *Behavior level*: When agents initiate conversations with their peers, they take different states based on the messages these agents submit and receive. It may occur that within certain states, the agents perform operations that initiate services. Thus, services take appropriate states.

3.3. Application of the three levels

After presenting the three-level specification approach, the current step consists of applying the approach to vacation scenario. Four services are required to handle this scenario. At present, the focus is on the specification of each service. The specification of agents happens once the agentification of services is completed (Section 4).

The application of the specification approach to vacation scenario occurs as follows (Fig. 5):

- A designer devises a C-service to be denoted by vacation C-service. The specification approach assists the designer in his work.
- The P-services that constitute vacation C-service are: flight reservation, hotel booking, attraction search, and user notification. These P-services are listed in component link property of the functional level of the service.
- Based on mandatory causal link property of the functional level of the service, *driving time calculation* and *car rental* P-services have to be added to vacation C-service (Fig. 5(a)). The first P-service checks the distance between the location of the hotel and the location of the main attraction. If the distance is greater to a user-defined limit, car rental P-service is triggered.
- Based on optional causal link property of the functional level of the service, *historic details* P-service can be added to vacation C-service (Fig. 5(b)). This P-service provides historic information on the city the user plans visiting. This new P-service is triggered upon the user's approval.

The application of service chart diagrams to vacation C-service occurs as follows. For the sake of space, only two primitive services are presented.

- Flight reservation P-service (Fig. 6(a)): it is the first P-service of vacation C-service to be triggered. It is followed by hotel booking and attraction search P-services. These P-services need for their processing departure date, return date, and city of destination to be obtained from flight reservation P-service. This latter P-service takes stand by and execution states.

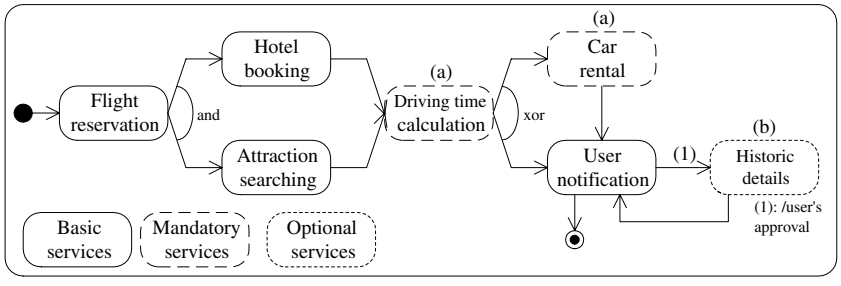

Fig. 5. Vacation composite-service.

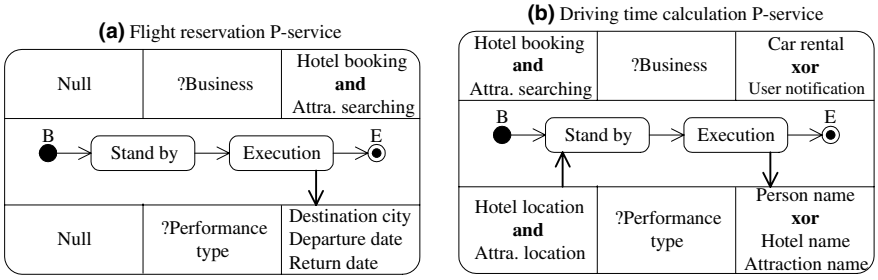

Fig. 6. Application of service chart diagram.

The businesses that will provide flight reservation P-service (?business attribute) and the invocation way of flight reservation P-service (?performance type attribute) are identified during run-time.

- Driving time calculation P-service (Fig. 6(b)): it has been added to vacation C-service according to causal link property of the functional level of the service. Hotel booking and attraction search P-services precede the execution of driving time calculation P-service. In addition, this P-service is followed either by car rental P-service or user notification P-service. The decision is made based on the distance that exists between the location of the hotel and the location of the attraction. According to the distance, driving time calculation P-service triggers the relevant P-service. Car rental P-service needs the name of the person for whom the lease of the car will be made. User notification P-service needs the hotel and attraction names so the user can be informed about all these details. The businesses that will provide driving time calculation P-service (?business attribute) and the invocation way of driving time calculation P-service (?performance type attribute) are identified during run-time.

4. Agentification of an environment of Web services

4.1. Architecture

For the agentification needs of an environment of Web services, we deployed an *agent-based multi-domain* architecture (Fig. 7). Domains are spread across the network and administrators maintain them. Two types of domain exist: *user-domain* and *provider-domain*. We assume the existence of one user-domain (despite the issue of bottleneck or a single point-of-failure that both could be handled with replication) and several provider-domains. Domains are computing platforms on which services and agents can run. Users browse the list of composite services from different devices whether fixed or mobile.

The user-domain has a *service-zone* and a *working-zone*. The service-zone has a list from which composite services are managed using the three-level specification approach. This list offers composite Web services to users of fixed devices and composite M-services to users of mobile devices. The service-zone of the user-domain has a bank from which user-agents are created. For their installation, user-agents are located in the working-zone of the user-domain. User-agents can migrate from one domain to another based on the strategy of invoking services. For each composite service that a user selects, a user-agent is associated with. We recall that only primitive Web services participate in composite services.

A provider-domain consists of a working-zone and several lists of primitive services. Each list is reserved to a specific category such as finance, education, and travel. The working-zones are devised in a way to receive user-agents arriving from the user-domain or from other provider-domains. Within provider-domains, installation and control procedures of user-agents are performed (these procedures do not fall within this paper's scope). Provider-agents handle the invocation requests that user-agents submit to the primitive services. A

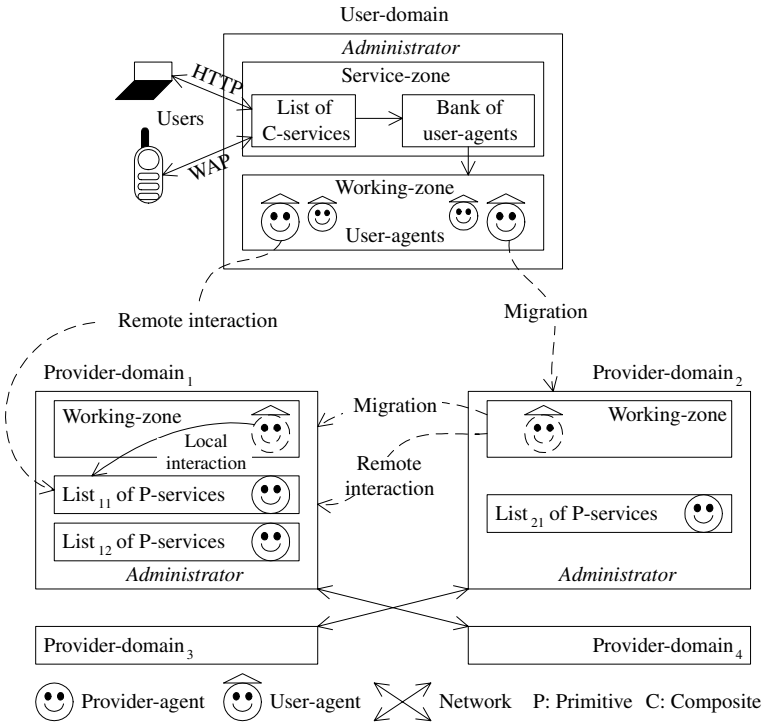

Fig. 7. Software agent-based multi-domain architecture.

user-agent submits a local request to a provider-agent in case both agents are in the same provider-domain. In case the user-agent and provider-agent are in separate domains, the user-agent submits a remote request to the provider-agent so, the primitive service can be triggered.

4.2. Operation

The operation of the multi-domain architecture consists of the specification of the composite services and their deployment once users initiate them. The specification of a composite service is discussed in Section 3.3. This means that the *communities* of component primitive-services of a composite service are already known (details on service communities are provided in [3]). Now, the focus is on deploying the composite service in terms of: (i) selecting the component services as providers can have services in common, and (ii) executing the component services selected as services can be remotely or locally invoked. Users browse the list of composite services from their fixed or mobile devices. The deployment of a composite service is not affected by the type of the device from which it is triggered. The only difference occurs at the communication protocol (HTTP vs. WAP) that connects users to the user-domain.

When a composite service is selected, the user indicates his needs (e.g., city of destination, number of persons). This leads into the creation of a user-agent to handle the satisfaction of the user's request. Initially, the user-agent is in the working zone of the user-domain. Then, the user-agent starts identifying the component primitive-services of the composite service. The user-agent relies on component link property of the functional level (Table 2). At this stage, the user-agent knows the appropriate component primitive-services. Before it interacts with the respective provider-agents of these primitive services, the user-agent checks if additional primitive services are not also required.

To this purpose, it checks mandatory causal link and optional causal link properties (Table 2). The primitive services that are identified in the mandatory causal link property have to be attached to the composite service. Regarding optional causal link property, the user-agent may decide to provide additional details that were not mentioned in the user's initial-request.

After the user-agent identifies the primitive services that *have* to be and *can* be attached to a composite service, it starts implementing the composite service by interacting with their provider-agents. The user-agent either migrates to a provider-domain in which it locally requests the execution of a primitive service. Or, the user-agent remotely requests the execution of a primitive service from the domain (either a user-domain or a different provider-domain) in which it currently resides. Once all the primitive services are executed, the user-agent returns details to the user. For the primitive services that were included according to optional causal link property, the user-agent checks whether the user is interested in getting extra information. The user may be informed about the charges related to the new primitive services.

4.3. Getting Web services ready

For a composite service, preparing the component primitive-services for composition and execution relies on two selection criteria: *execution cost* and *location of computing hosts* (a computing host corresponds to a provider-agent). Execution cost criterion is directly related to a primitive service. Whereas the location of computing host criterion aims at gathering in the same provider-domain the *maximum* number of primitive services for execution, privileging local interactions over remote interactions. This means reducing: (i) the number of remote interactions between domains, (ii) the number of migrations of user-agents to provider-domains, and (iii) the number of remote data exchanges between domains.

The identification of the provider-domains and their computing host is based on an Algorithm that is given in Fig. 8. First, the domain of where a service is currently being executed is considered (set *A* in Fig. 8, line 03). By selecting this domain, remote data exchanges between services are avoided. Next, the domain of where the user-agent currently resides is considered (set *B* in Fig. 8, line 04). By selecting this domain, local invocations of

```
01: for each < p.s_i, PRO.AGT_i, type >, i = 2, ..., p
02: begin
03:   | A ← φ //provider-agents in the same domain as pro.agt_{i-1}
04:   | B ← φ //provider-agents in the same domain as user-agent
05:   | C ← φ //provider-agents that are in other domains
06:   | for (j = 1; j <= ||PRO.AGT_i||; j++) //pro.agt_j ∈ PRO.AGT_i
07:   | begin
08:   |    | if domain(pro.agt_j) = domain(pro.agt_{i-1})
09:   |    | then A ← A ∪ pro.agt_j
10:   |    | else if domain(pro.agt_j) = domain(user − agent)
11:   |    |      then B ← B ∪ pro.agt_j
12:   |    |      else C ← C ∪ pro.agt_j
13:   | end //A ∪ B ∪ C = PRO.AGT_i
14:   | if A ≠ φ
15:   | then contact provider-agents of A - Go Phase 2.1
16:   | else if B ≠ φ
17:   |     then contact provider-agents of B - Go Phase 2.1
18:   |     else contact provider-agents of C - Go Phase 2.1
19: end
```

Fig. 8. Algorithm for selecting provider-agents.

services as well as local data exchanges between services are enabled. In case none of the aforementioned cases happen, any domain is considered (set C in Fig. 8, line 05).

4.3.1. Definitions

A user-agent expects to: (i) associate each primitive service with a provider-agent and (ii) define the strategy of invoking the primitive service. We assume a C-service CS of n P-services, $\mathrm{CS} = \{p.s_1, p.s_2, \ldots, p.s_n\}$. The specification of CS corresponds to the set $\{< p.s_1, \mathrm{pro.agt}_1, type >, < p.s_2, \mathrm{pro.agt}_2, type >, \ldots, < p.s_n, \mathrm{pro.agt}_n, type >\}$ where $\bigcup_{i=1}^{n} p.s_i = \mathrm{CS}$ and for each $< p.s_i, \mathrm{pro.agt}_i, type >_{(i\in[1,n])}$ the P-service $p.s_i$ is provided by the provider-agent $\mathrm{pro.agt}_i$ and invoked according to remote or local *type*. The number of provider-agents that contribute to the provisioning of CS is not necessarily equal to the number of P-services that are involved in a composite service. Certain provider-agents may contribute with more than one P-service (e.g., $< p.s_1, \mathrm{pro.agt}_1, type >$ and $< p.s_2, \mathrm{pro.agt}_1, type >$).

Given a P-service, its execution cost is decomposed into two parts:

- Remote cost of a P-service includes: (i) the cost of establishing a communication link between the domain in which the user-agent is now located and the provider-domain of the provider-agent of the P-service, plus (ii) the cost of performing the P-service, plus (iii) the cost of sending back the results from the provider-domain of the provider-agent to the domain of the user-agent.
- Local cost of a P-service includes: (i) the cost of moving the user-agent from the domain in which it now resides to the provider-domain of the provider-agent of the P-service, plus (ii) the cost of performing the P-service.

4.3.2. Preparation

The preparation of the primitive services for composition is divided into two phases. Phase 1 consists of searching for the provider-agents that have the P-services. Because provider-agents can have P-services in common, Phase 2 consists of selecting a particular provider-agent based on the criteria of execution cost and location of computing hosts.

In Phase 1, the identification of the provider-agents is based on the business perspective of the service chart diagram (Fig. 2). For each P-service $p.s_i$, potential provider-agents are identified. This is similar to $< p.s_i, \mathrm{PRO.AGT}_i, type >$ where $\mathrm{PRO.AGT}_i = \{\mathrm{pro.agt}_1, \ldots, \mathrm{pro.agt}_m\}$ is the list of provider-agents that have the P-service $p.s_i$ in common, for example $< p.s_1, \mathrm{PRO.AGT}_1, type >$, where $\mathrm{PRO.AGT}_1 = \{\mathrm{pro.agt}_1, \mathrm{pro.agt}_2, \mathrm{pro.agt}_3\}$.

In Phase 2, because provider-agents can have P-services in common, the association of a P-service with a specific provider-agent has to be completed. In addition, because of the location criterion the P-services are treated one at a time. The definition of $< p.s_i, \mathrm{pro.agt}_i, type >$ is broken down into two sub-phases.

In Phase 2.1, the P-service $p.s_{i\,(i=1)}$ of CS is considered. At this level, only the execution-cost selection-criterion is considered (location criterion does not hold). From each provider-agent of $\mathrm{PRO.AGT}_{i\,(i=1)}$ that offers the P-service $p.s_{i\,(i=1)}$, the user-agent receives the execution cost for remote and local invocation types (Eq. (1)).

User-agent :

$$p.s_{i\,(i=1)} \begin{cases} \mathrm{pro.agt}_1 : (remoteCost(p.s_i^1), localCost(p.s_i^1)) \\ \vdots \\ \mathrm{pro.agt}_k : (remoteCost(p.s_i^k), localCost(p.s_i^k)) \end{cases} \tag{1}$$

For each offer that a provider-agent $\mathrm{pro.agt}_k$ submits to the user-agent, the user-agent selects the minimum cost between the two types of invocation $(\min(remoteCost(p.s_i^k), localCost(p.s_i^k)))$. Afterwards, the user-agent selects for the P-service $p.s_{i\,(i=1)}$ the minimum cost among all the offers of the provider-agents. For example, the user-agent sets $< p.s_1, \mathrm{pro.agt}_2, remote >$: $\mathrm{pro.agt}_2$ provides $p.s_1$ and $p.s_1$ is remotely invoked. Because of the remote invocation, the user-agent and $pro.agt_2$ will definitely be in two different domains.

In Phase 2.2, the user-agent continues with the remaining P-services $p.s_i, (i = 2, \ldots, n)$ also one at a time. Now, the two selection criteria are simultaneously considered. It should be noted that the location criterion is privileged over the execution

cost criterion due to the aforementioned benefits. Considering the location criterion, the provider-agent that is associated with a P-service $p.s_{i\ (i\in[2,n])}$ depends on the provider-agent that has been selected to offer the predecessor P-service $p.s_{i-1}$. The user-agent proceeds according to the algorithm of Fig. 8 (in the algorithm, $< p.s_{i-1}, \text{pro.agt}_{i-1}, local|remote >$ is assumed). When the user-agent finishes working on a P-service $p.s_i$, its provider-agent and invocation strategy of that P-service are known. The purpose of the algorithm is to generate a short list of provider-agents with whom the user-agent will interact about their services. This short list ranks the domains in which the provider-agents and user-agent reside (A, B, and C sets, lines 08–12).

4.3.3. Execution

When all the P-services of a C-service are identified, the user-agent starts invoking these P-services through their provider-agent. For illustration, the following C-service is used $\text{CS} = \{< p.s_1, \text{pro.agt}_1, local >, < p.s_2, \text{pro.agt}_2, local >, < p.s_3, \text{pro.agt}_3, remote >\}$. Initially, the user-agent is in the user-domain, pro.agt_1 and pro.agt_2 are both in provider-domain_1, and pro.agt_3 is in provider-domain_3. Since $p.s_1$ is going to be locally executed, the user-agent migrates from the user-domain to provider-domain_1. Once the execution of $p.s_1$ is completed, the user-agent locally executes $p.s_2$. Because pro.agt_2 is within the same domain as pro.agt_1, this shows an implementation of the location criterion. This also shows that the transfer of data from $p.s_1$ to $p.s_2$ if both are interdependent is locally done, which avoids dealing with network-connection failures. Finally, from provider-domain_1 the user-agent submits a remote request to pro.agt_3 so $p.s_3$ can be executed. The transfer of data from $p.s_2$ to $p.s_3$ if both are interdependent is remotely done, which means the importance of being aware of network-connection failures.

5. Work in-progress

Several aspects are under development in the agent-based multi-domain architecture of Fig. 7. In this paper, we discuss three of them namely interleaving Web services composition and execution, reliability of composite services execution, and prototype implementation.

5.1. Interleaving Web services composition and execution

It is shown in Section 4.3 that the deployment of a composite service is sequential and demands from the user-agent: (i) to process all the provider-agents' offers about the primitive services, (ii) to constitute the composite service, and finally (iii) to execute the composite service. It would be more appropriate if the composition and execution of services could be interleaved [18]. The user-agent has to delegate a part of its work (either composition or execution) to a third party. To this end, a delegate-agent is concurrently created to the user-agent deployment. While the user-agent is remotely interacting with provider-agents or visiting domains of provider-agents, the delegate-agent is preparing the component services for execution on behalf of the user-agent, and submitting the details on what the user-agent has to carry out. These details concern the provider-agents of the component services and the invocation types of these component services. While the delegate-agent is working on the P-service $p.s_i$, the user-agent is executing the P-service $p.s_{i-1}$. Therefore, the delegate-agent is always one-step ahead of the user-agent. User-agent and delegate-agent communicate in two ways: locally when both agents are in the user-domain and remotely when the delegate-agent is in the user-domain and the user-agent is in one of the provider-domains visiting their provider-agents.

Fig. 9 illustrates the way user-agent and delegate-agent implement the interleaving of Web services composition and execution. We assume a C-service $\text{CS} = \{< p.s_1, ?\text{pro.agt}, ?type >, < p.s_2, ?\text{pro.agt}, ?type >\}$. First of all, the delegate-agent starts working on $p.s_1$ for execution. Based on the offers it receives from provider-agents, the delegate-agent establishes for $p.s_1$ what follows: local execution in provider-domain_1 of pro.agt_1 ($< p.s_1, \text{pro.agt}_1, local >$). Therefore, the delegate-agent asks the user-agent to migrate to provider-domain_1. Before it moves, the user-agent resides in

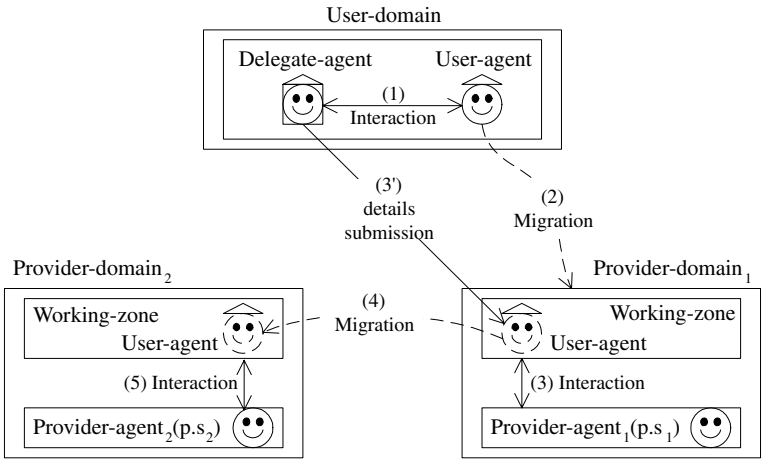

Fig. 9. Interleaving service composition and execution.

the user-domain. While the user-agent is getting ready for migration, the delegate-agent starts the preparation of $p.s_2$. After performing all the necessary operations, the delegate-agent establishes for $p.s_2$ what follows: local execution in provider-domain$_2$ of pro.agt$_2$ ($< p.s_2, \text{pro.agt}_2, \textit{remote} >$). The details on $p.s_2$ are transferred to the user-agent that is now located in provider-domain$_1$. After the user-agent finishes the execution of $p.s_1$, it moves to provider-domain$_2$ to locally interact with pro.agt$_2$.

5.2. Reliability of composite services execution

The reliability of a Web service is defined as the probability that a request submitted to a Web service is correctly responded within the maximum expended time frame [24]. This time frame is mostly published as part of the Web service description. Reliability is a technical measure that depends on hardware and/or software configuration of Web services and on network connections between requestors and service providers. The reliability value can be computed from historical data about past invocations using for example the number of times that a Web service has been successfully delivered within the maximum expected time frame, with regard to the total number of invocations.

Because reliability deals with service execution failures, backup approaches are deemed appropriate. A Web service cannot be executed for multiple reasons: network connection problems, service disconnected for maintenance, service overloaded, just to cite a few. In the following, we present the way reliability is integrated into the operation of the multi-domain architecture of Fig. 7.

Interleaving Web services composition and execution has called for two types of agents: user-agent and delegate-agent. Initially, the delegate-agent associates a Web service with a provider-agent and submits that information to the user-agent that must be running either in the user-domain or in one of the multiple provider-domains. The selected provider-agent is part of a pool of potential provider-agents (PRO.AGT$_i$) that have a Web service in common. Before the delegate-agent starts working on the next component services, it stores the information about the pool of provider-agents (for example the "x" best ranked provider-agents from sets A, B, and C of PRO.AGT$_i$, $x <= (\|\text{PRO.AGT}_i\| - 1)$) for a later use. If the user-agent faces any difficulties in the execution of a service, it immediately contacts the delegate-agent which is always located in the user-domain. Because the delegate-agent is now working on the preparation of the remaining

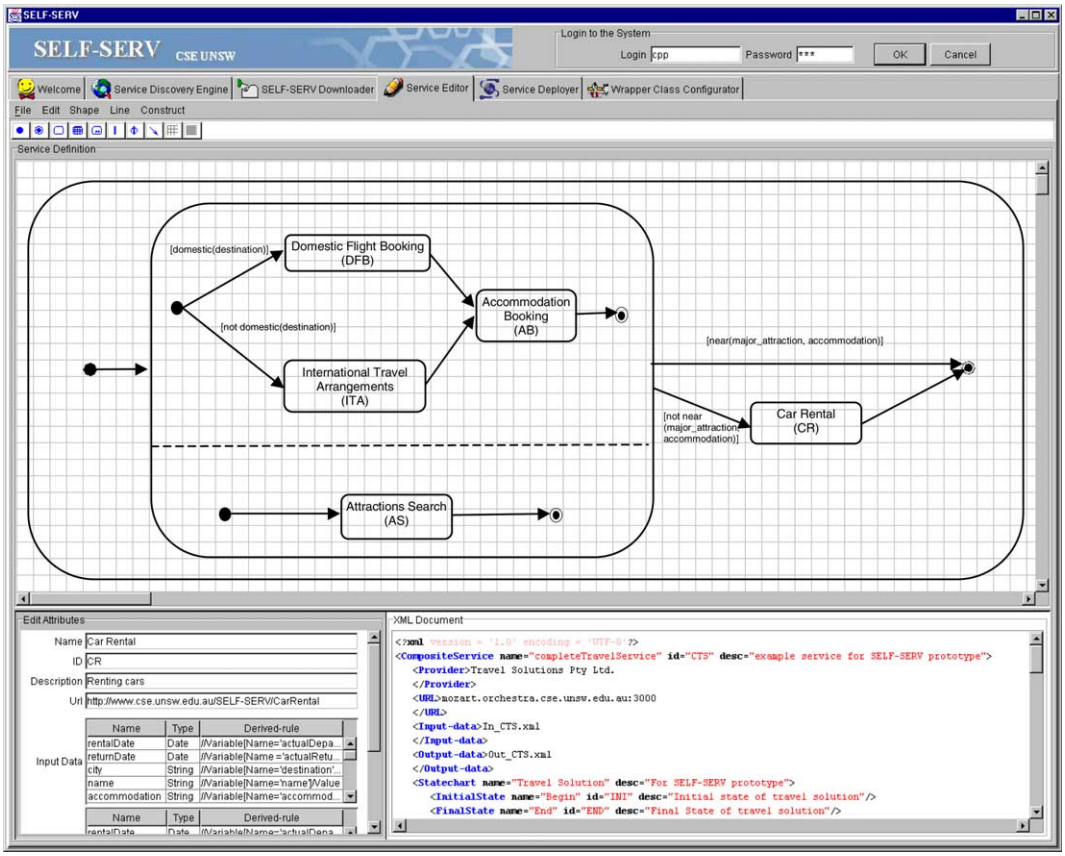

Fig. 10. Web services composition.

component services, it stops its preparation work and browses the stored pool of provider-agents for the service in trouble. The objective is to identify a new provider-agent, [3] inform the user-agent about this provider-agent, and finally store the newly updated pool of potential provider-agents. Information on a pool of potential provider-agents are not deleted until the delegate-agent receives a notification message from the user-agent that the execution of a service has been successfully completed. During that confirmation exchange, the delegate-agent submits to the user-agent the details on the next service to execute.

[3] The delegate-agent may request new offers from the provider-agents that are stored in the pool.

5.3. Implementation of prototype

We overview the status of the prototype implementation. The prototype architecture consists of a *service composition environment* and a *pool of services and agents*. All these components are being implemented in Java, whereas services communicate through XML documents.

5.3.1. Service composition environment

The service composition environment consists of a set of integrated tools that allow service providers and users to create and execute services. WSDL is used to specify Web services and UDDI is used as a service repository. Since WSDL focuses on how to invoke a Web service, some of the attributes proposed in our approach are not supported by WSDL (e.g., service domain). To over-

come this limitation, such attributes are specified as tModels. Each tModel represents the specification of one attribute. The keys of these tModels are included into the `categoryBag` of the tModel of a Web service.

The service builder assists providers in defining new services and editing existing ones as well. A service definition is edited through a visual interface (Fig. 10), and translated into an XML document for further processing. The service builder offers an editor for describing a service chart diagram of a component service participating in a composite service (an extension of our previous work in [19]). It also provides means to describe the properties of states (e.g., state ID, state name, component service operation) and transitions (e.g., ECA rules).

5.3.2. *Pre-built agents*

For any user (resp., service) wishing to participate in our platform, the user (resp., the administrator of the service) needs to download and install a set of pre-built agents, namely *user agent* (resp., *provider-agent*). The functionalities of the provider agents are realized by a class called *serviceWrapper*, which provides methods for receiving, processing, generating and sending control-flow notifications, service invocation, and service completion messages.

User agents are mobile agents implemented using Aglet [1]. IBM's Aglets Software Development Kit V2 is used for implementing Aglets. The only infrastructure required to install and configure these agents are Java, Tahiti (a tiny Aglet server program) and an XML parser. The functionalities of user agents are realized by a class called *userAgent*. Upon the initialization of the `userAgent`, a *delegateAgent* is created and installed in the user domain, which is responsible for: (i) preparing the execution plan of each component of a composite service; and (ii) submitting details on the execution plan to `userAgent`.

The `userAgent` is a static Aglet. When a service needs to be locally invoked, `userAgent` creates a slave called *userAgentSlave*, and passes the destination (e.g., service host) to the `userAgentSlave`. `userAgentSlave` is the labor Aglet that actually goes to the service site. Upon

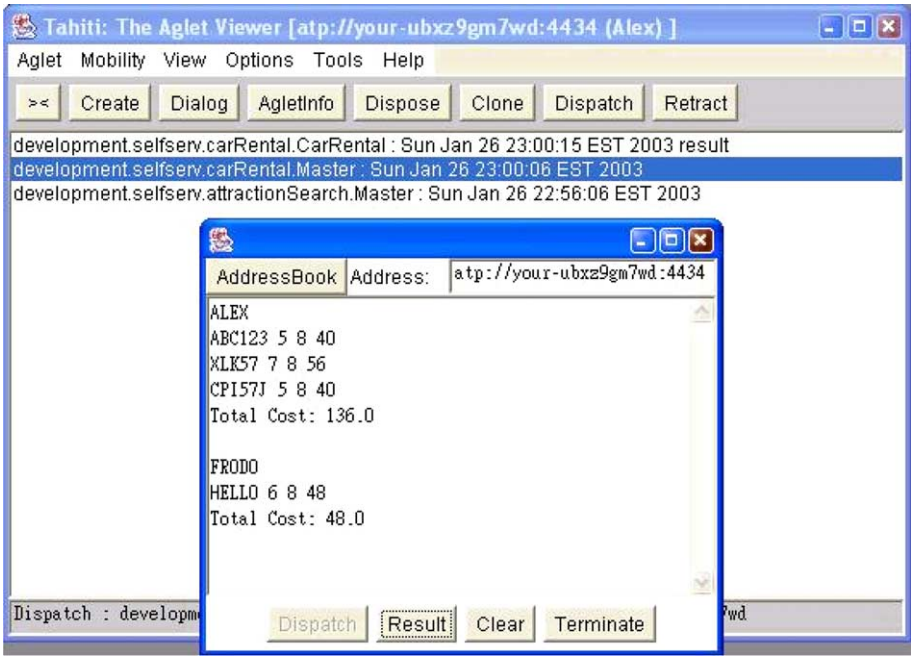

Fig. 11. Service execution using user agent.

arrival at the service site, `doJob` method of the `userAgentSlave` is called, which performs the real work that we assign to the slave (i.e., invoke the service). Depending on the invocation type (i.e., locally or remotely) of next service, `userAgentSlave` either migrates to the site of the service or sends a request message for invocation needs. When all the component services are invoked, the `userAgentSlave` returns to the user-domain, `callBack` method of the master Aglet `userAgent` is activated. The results are passed as an argument of the `callBack` method. The `userAgent` Aglet then extracts and passes the results to the user. Fig. 11 shows the screenshot for service execution using user agent. The Aglet viewer window displays and controls the Aglets. When the button `Dispatch` is clicked, a slave agent is created for the `userAgent` and migrates to the destination for the execution of services. The results are returned to the `userAgent` and displayed in the Aglet viewer window. The detailed results can be obtained by simply clicking on the result message (e.g., `carRental` in the figure).

6. Related work

Several research projects have studied Web services composition [2]. With the progress of wireless technologies and handheld devices, mobile services are attracting the attention of both academia and industry [4,7,17]. As example, the MObile Teamwork Infrastructure for Organization Networking (MOTION) addresses the mobile teamwork requirements of organizations in their daily business [10]. MOTION provides support to mobile teamwork such as locating distributed business documents and expertise through peer-to-peer searches, advanced subscription and notification, community building, and mobile information sharing and access.

The importance of enhancing agents with mobility mechanisms has been pointed out in [22]. Wang worked on a system that allows the dispatch of multiple mobile agents in parallel when visiting e-shops. The mobile agent approach is suitable for deploying parallel processes over distributed sites on the Internet. The tasks to undertake are decomposed and encapsulated into multiple mobile agents. In the framework that Wang suggests, a mobile agent service provider is proposed as an execution environment for mobile agents [22]. This environment is similar to the working zone of the multi-domain architecture where creation, installation, verification, and performance operations are conducted.

Chakraborty et al. [6] have introduced a reactive service composition architecture for pervasive computing environments. The architecture consists of five layers: network, service discovery, service composition, service execution, and application. While reviewing Chakraborty et al.'s work, we were interested in the service execution layer. During the execution of services, this layer might want to optimize the bandwidth required to transfer data over the wireless links between services and hence, execute the services in an order that minimizes the bandwidth utilization. This optimization approach is similar to the location criterion that we introduced. With that criterion, we reduced the number of remote interactions between domains, the number of migrations of the user-agent to domains, and the number of data transfer between domains.

Another relevant work to the location criterion is presented in [16]. Because it will be challenging to create services that can execute well on the large variety of devices (problems of diversity and resource constraints), Messer et al. [16] suggest to transparently offload portions of a service code from resource-constrained devices to nearby servers. Code offloading requires *partitioning* strategies. If two components interact frequently (e.g., because of many method invocations), then a partitioning strategy should suggest placing these components together on one machine; splitting them across the network could severely affect performance. The aforementioned partitioning strategy has similarities with the location of computing hosts criterion. This criterion promotes the use of local interactions between services as well as between agents. In [16], the selection of the same host may cause an overloading for that host. In this paper, this situation is avoided for two main reasons. First, the work is done at the level of domains of computing hosts rather than at the

level of computing hosts. Second, the location criterion helps in finding domains (Fig. 8). When a domain is considered, traditional selection criterion (such as execution cost and execution time) are applied to identify the best computing hosts and thus, the best providers of services.

7. Conclusion

In this paper, we presented a specification approach for Web services composition and deployment. The approach has been featured by the use of different types of software agents and three levels of specification. The levels were denoted by intrinsic, organizational/functional, and behavior, and illustrated with a running scenario. Because of the complexity of Web services composition and deployment, it has been suggested to associate users with software agents and associate component services with service chart diagrams. A service chart diagram represents a service from five perspectives: state, flow, business, information, and performance.

In this paper, the composition and deployment of Web services have been handled by a two-phase process. The first phase has consisted of identifying the providers that offer the Web services that participate in composite services. The second phase has consisted of selecting specific providers according to two criteria: execution cost, and location of computing hosts. Location criterion has been privileged over the first criterion, enabling for instance to avoid cross-network traffic.

Our ongoing work includes the assessment of the performance and scalability of the proposed agent-based multi-domain architecture. We also plan to focus on interleaving Web services composition and execution. It was shown from the basic example of Fig. 9 that interleaving presents benefits when it comes to considering the context in which Web services evolve.

Acknowledgements

The authors would like to thank the referees for their valuable comments and suggestions of improvements. The authors also acknowledge the implementation work of Aysha Alsayed Almarzouqi, Alex Yue-Fai Tang, Eileen Oi-Yan Mak, and Nathan Wong. The third author's work was in part supported by an Australian Research Council (ARC) Discovery grant #DP0211207.

References

[1] Aglet. http://www.trl.ibm.com/aglets/, Visited September 2003.
[2] B. Benatallah, F. Casati (Guest Editors), Special issue on Web services, Distributed Parallel Databases 12 (2–3) (2002).
[3] B. Benatallah, Q.Z. Sheng, M. Dumas, The self-serv environment for web services composition, IEEE Internet Computing 7 (1) (2003).
[4] G. Caire, N. Lhuillier, G. Rimassa, A communication protocol for agents on handheld devices, in: Proceedings of the First International Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices held in conjunction with the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2002), Bologna, Italy, 2002.
[5] D. Chakraborty, A. Joshi, Dynamic service composition: State-of-the-art and research directions, Technical report, TR-CS-01-19, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, MD, USA, 2001.
[6] D. Chakraborty, F. Perich, A. Joshi, T. Finin, Y. Yesha, A reactive service composition architecture for pervasive computing environments, in: Proceedings of the Seventh Personal Wireless Communcations Conference (PCW'2002), Singapore, 2002.
[7] I. Chisalita, N. Shahmehri, Issues in image utilization with mobile e-services, in: Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'2001), Cambridge, MA, USA, 2001.
[8] J.Y. Chung, K.J. Lin, R.G. Mathieu, Web services computing: advancing software interoperability, IEEE Computer 36 (10) (2003).
[9] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, S. Weerawarana, The next step in web services, Commun. ACM 46 (10) (2003).
[10] P. Fenkam, E. Kirda, S. Dustdar, H. Gall, G. Reif, Evaluation of a publish/subscribe system for collaborative and mobile working, in: Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'2002), Pittsburgh, Pennsylvania, USA, 2002.
[11] D. Harel, A. Naamad, The STATEMATE semantics of statecharts, ACM Trans. Software Eng. Methodol. 5 (4) (1996).

[12] N. Jennings, K. Sycara, M. Wooldridge, A roadmap of agent research and development, Autonomous Agents Multi-Agent Systems 1 (1) (1998).
[13] Z. Maamar, Moving code (Servlet Strategy) vs. inviting code (Applet Strategy) – which strategy to suggest to software agents?, in: Proceedings of the Third International Conference on Enterprise Information Systems (ICEIS'2001), Setubal, Portugal, 2001.
[14] Z. Maamar, B. Benatallah, W. Mansoor, Service chart diagrams – description and application, in: Proceedings of the 12th International World Wide Web Conference (WWW'2003), Budapest, Hungary, 2003.
[15] Z. Maamar, W. Mansoor, Design and development of a software agent-based and mobile service-oriented environment, e-Service J. 2 (3) (2003).
[16] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T.J. Giuli, X. Gu, Towards a distributed platform for resource-constrained devices, in: Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'2002), Vienna, Austria, 2002.
[17] D. Milojicic, A. Messer, P. Bernadat, I. Greenberg, G. Fu, O. Spinczyk, D. Beuche, W. Schroder-Preikschart, Ψ's – pervasive services infrastructure, Technical Report HPL-2001-87, HHP Laboratories, Palo Alto, CA, USA, 2001.
[18] M. Paolucci, O. Shehory, K. Sycara, Interleaving Planning and Execution in a Multiagent Team Planning Environment, Technical report, CMU-RI-TR-00-01, The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA, 2000.
[19] Q.Z. Sheng, B. Benatallah, M. Dumas, E. Mak, SELF-SERV: a platform for rapid composition of web services in a peer-to-peer environment, in: Proceedings of the 28th Very Large DataBase Conference (VLDB'2002), Hong Kong, China, 2002.
[20] M.P. Singh, The pragmatic web, IEEE Internet Comput. 6 (3) (2002).
[21] A. Tsalgatidou, T. Pilioura, An overview of standards and related technology in web services, Distributed Parallel Databases 12 (2–3) (2002) 135–162.
[22] Y. Wang, Dispatching multiple mobile agents in parallel for visiting e-shops, in: Proccedings of the Third International Conference on Mobile Data Management (MDM'2002), Singapore, 2002.
[23] J. Yang, M. Papazoglou, W.-J. van den Heuvel, Tackling the challenges of service composition in e-marketplace, in: Proceedings of the 12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce/E-Business Systems (RIDE-2EC'2002) in Conjunction with ICDE'02, San Jose, USA, 2002.
[24] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, Q.Z. Sheng, Quality driven Web service composition, in: Proceedings of the 12th International World Wide Web Conference (WWW'2003), Budapest, Hungary, 2003.