



On Composite Web Services Provisioning in an Environment of Fixed and Mobile Computing Resources

ZAKARIA MAAMAR *

zakaria.maamar@zu.ac.ae

College of Information Systems, Zayed University, United Arab Emirates

QUAN Z. SHENG and BOUALEM BENATALLAH

{qsheng,boualem}@cse.unsw.edu.au

School of Computer Science & Engineering, The University of New South Wales, Australia

Abstract. We present a framework for Web services provisioning in a hybrid environment of fixed and mobile computing resources. Several obstacles still hinder the seamless provisioning of Web services in mobile environments. Examples of such obstacles are: throughput and connectivity of wireless networks, limited computing resources of mobile devices, and risks of communication channel disconnections. In the proposed framework, software agents represent users, providers of services, and providers of resources. The business logic of composite services is expressed as a process model using statecharts formalism. Among other things, the use of agents provides an infrastructure that has the ability to handle disconnections during service preparation for execution. The framework also integrates a service execution planning approach to optimally select computing resources (fixed or mobile) on top of which services will be executed.

Keywords: Web services, fixed/mobile resources, service provisioning, process-based service composition, software agents

1. Introduction

The explosive growth of interconnected computing technologies (e.g., mobile phones, Bluetooth, I-mode) enables new environments where ubiquitous information access will be a reality. More importantly, Web services will be accessible from mobile devices [27].

With the current progress of telecommunication technologies, new Web services are being offered to users of mobile devices, e.g., cellular phones and personal digital assistants. M-services (M for mobile) denote these Web services [18]. It happens that a user has to postpone her operations because she lacks appropriate facilities on her mobile device, e.g., an application that converts a drawing file into a format that the user's mobile device can display. To avoid such a situation, a user should be able to search for additional facilities when needed and either (i) invoke these facilities remotely or (ii) download these facilities from a web site to her mobile device. From a user perspective, it is important to make sure that all these operations are carried out in a transparent way. Therefore, Software Agents (SAs) are deemed appropriate to achieve

* Address for correspondence: P.O. Box 19282, Dubai, United Arab Emirates.

that transparency [19]. Despite the multiple opportunities that M-services can offer to users (especially those who are on the move, e.g., sales representatives), different obstacles still hinder the expansion of M-services, e.g., mobile devices are still limited by the fact that they are battery powered. Because users are eager to operate more services from their mobile devices, telecommunication companies are making tremendous effort to offer new local and wide-area wireless networks and as well new mobile devices [23,28]. Evidences of these efforts are multiple, e.g., 3G systems [11] and I-mode technology [22].

Web services provisioning is a very active area of research and development [3]. However, *very little* has been done regarding the provisioning of services in hybrid environments where services may be hosted on mobile or fixed computing resources [18]. In particular, several obstacles still hinder the seamless provisioning of Web services in mobile environments. Examples of such obstacles are: throughput and connectivity of wireless networks, limited computing resources of mobile devices, and risks of communication channel disconnections. To optimize service provisioning in mobile environments, several important issues need to be considered:

- *Handling disconnections during service execution*: In a mobile environment, disconnections may be frequent (e.g., disconnection due to the problems of battery or communication costs, disconnection due to the fact that devices can change location). As a consequence, dealing with user or service device disconnection during service execution is a critical issue.
- *Context-sensitive service execution planning*: In addition to criteria such as monetary cost and execution time, service execution planning should consider the location of requesters of services, capabilities of computing resources on which services will be executed (e.g., CPU, bandwidth), and so on. It is a necessity to enable the system to adapt itself to different computing and user's requirements. Therefore, the identification of the resources on which the execution of the services takes place is important.

In this paper, we present an agent-based architecture for Web services provisioning in hybrid environments. In this architecture, users and services are represented by SAs. In a nutshell, SAs are autonomous entities that act on behalf of users, make decisions, interact with other agents, and roam the net if needed [19]. An agent-based architecture offers interesting features for service provisioning in hybrid environments. For example, an agent may be used to collect execution results during disconnection of the user's device, and returns these results to the user upon re-connection.

The business logic of a composite Web service is expressed as a process model that uses statecharts [24]. Statecharts formalism [13] is widely accepted as a suitable language for modeling complex and large workflow applications [29]. The process model underlying a composite service identifies the functionalities required by the component services and their interactions (i.e., control-flow, data-flow). Actual services that are able to provide the required functionalities are selected during each execution of the composite service. Based on a set of service quality criteria (e.g., execution cost, resource relia-

bility), we propose a service execution planning approach that can be used to optimally select computing resources on which (fixed or mobile) services will be executed.

In this paper, we focus on: (i) *agent support* and (ii) *optimal resources selection* for service provisioning in hybrid environments. Section 2 presents the concepts of software agents, mobile computing, and Web services. Section 3 presents the agent-based architecture for service provisioning in hybrid environments. Section 4 presents the service execution planning approach in such environments. Section 5 discusses the implementation of the proposed framework. Section 6 discusses related work. Finally, section 7 concludes the paper and presents our future work.

2. Background

2.1. Software agent

An SA is a piece of software that acts autonomously to undertake tasks on users' behalf [15]. The design of many SAs 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. An SA exhibits a number of features that make it different from other traditional components including autonomy, goal-orientation, collaboration, flexibility, self-starting, temporal continuity, character, communication, adaptation, and mobility.

2.2. Mobile computing

Mobile computing refers to systems in which computational components, either hardware or software, change locations in a physical environment [14]. Bellavista et al. talk about user mobility, terminal mobility, and mobile access [2]. User mobility requires providing users with a uniform view of their preferred working environment independent of their current position in the network. Terminal mobility allows devices to transparently move and connect to different points of attachment. Mobile access is an emerging issue that involves the dynamic adaptation of mobile-aware resources and services. Mobile users and terminals can automatically retrieve such resources/services regardless of their current location.

2.3. Web services

A Web service is an accessible application that can be automatically discovered and invoked by other applications (and humans). We adopt a definition which considers an application as a Web service if it is: (i) independent as much as possible from specific platforms and computing paradigms; (ii) developed mainly for inter-organizational situations rather than for intra-organizational situations; and (iii) easily composable (i.e., its composition with other Web services does not require the development of complex adapters) [5]. In general, a Web service is specified by an identifier, a set of attributes,

and a set of operations. The attributes of a service provide information which is useful to the service's potential consumers, e.g., public key certificates.

Maamar et al. introduce the concept of M-services (M for Mobile) as a specific type of Web services [17]. Two definitions are suggested. The *weak* definition is to remotely trigger a Web service from a mobile device for execution. In this case, the Web service is an M-service. The *strong* definition is to transfer a Web service from its hosting site to a mobile device where its execution takes place. In this case, the Web service is 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;¹ and finally (iv) runnable on mobile devices. In this paper, we focus on the M-services that comply with the strong definition. Indeed, we are mainly interested in devising Web services that can be provisioned to users of mobile applications. Submitting services to be run on mobile devices has been supported by [8].

In the rest of this paper, the term composite service denotes the list of component services, whether Web services or M-services, that are involved in a composition. In addition, in order to ensure a wide applicability of our work, the suggested concepts are intended to be independent of specific service implementation technologies (e.g., .Net, Java), service description languages (e.g., WSDL), and service registration and discovery infrastructure (e.g., UDDI).

3. Agent-based architecture for service provisioning in hybrid environments

Figure 1 is the agent-based architecture that supports the service composition approach. The architecture consists of three parts. The first part corresponds to providers having services (S) or resources (R) to offer. The second part corresponds to users who play a dual role: provider of resources (R) when they advertise their fixed or mobile devices as computing resources, and consumers of services when they trigger the execution of services. Finally, the third part is a meeting platform on top of which brokers undertake the necessary matching operations [16]. The platform is the link between the provider and user parts. Because of the meeting platform, all the interactions between users and providers take place locally. There is no longer need for remote interactions. These interactions are related first to the identification of the component services based on the user's selection and second to the identification of the computing resources on which these component services will be executed. In figure 1, each part of the architecture is associated with multiple types of agents. In the same figure, numbers in parenthesis correspond to the chronology of operations in the agent-based architecture. This chronology is well detailed in figure 2.

Provider-agents are broken down into two types: resource provider agent and service provider agent (providers having services and resources at the same time are associated with both types of agents). Provider-agents are represented in the meeting

¹ The 3W Composite Capabilities/Preferences Profiles (CC/PP) working group aims at putting forward standards that will enable providers of services to get the capabilities of devices through their profile.

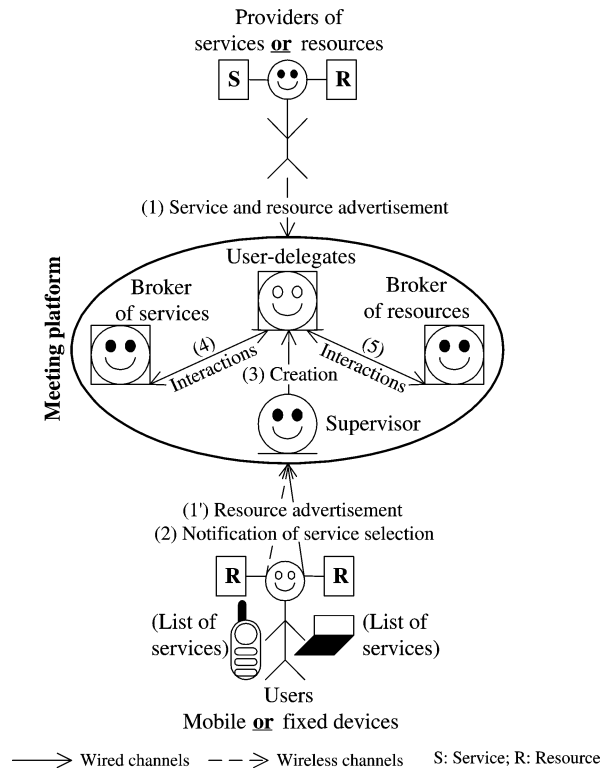


Figure 1. Agent-based architecture of the service composition approach.

platform by delegates, denoted respectively as service-delegates and resource-delegates. To keep figure 1 clear, both delegates are not represented. It should be noted that delegates are agents. But, for the sake of simplicity the word delegate is used instead of agent.

User-agents run on top of users' devices whether fixed or mobile. Since a user plays a double role, the user-agent assists the user in (i) advertising the computing resources of her devices to the resource-broker-agent (which means users will have resource-delegates in the meeting platform) and (ii) submitting the information about the services the user has selected for triggering to the supervisor-agent (figure 1, interaction (2)). The supervisor-agent will have to create user-delegates in the meeting platform (figure 1, interaction (3)).

Broker-agents are specialized into two types: service broker agent and resource broker agent. The service-broker-agent receives notifications from service-delegates regarding the services that service-provider-agents offer (figure 1, interaction (1)). In addition, the service-broker-agent receives requests from user-delegates regarding the services they need to be involved based on the services that users have triggered (figure 1, interaction (4)). Meanwhile, the resource-broker-agent receives notification messages from resource-delegates regarding the resources that resource-

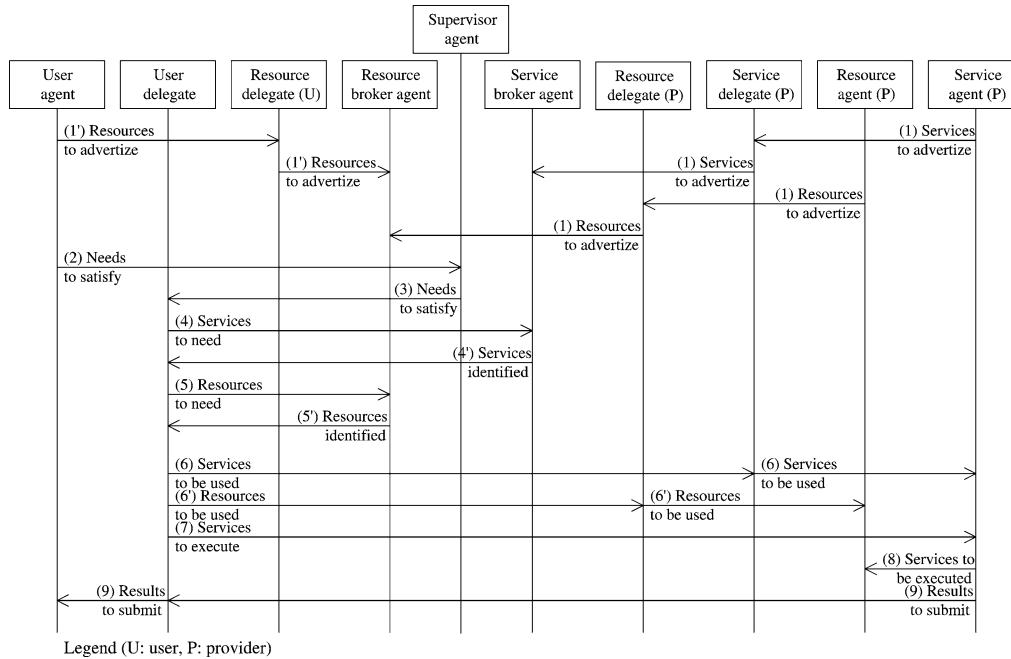


Figure 2. Interaction diagram of the agent-based architecture.

provider-agents and user-agents have (figure 1, interactions (1) and (1')). In addition, the resource-broker-agent receives requests from user-delegates regarding the resources that are needed for the execution of the component services that user-delegates have obtained from the service-broker-agent (figure 1, interaction (5)). *It should be noted that the interactions between user-delegates, resource-broker-agents, and resource-delegates are the focus of this paper.* Other interactions between user-delegates, service-broker-agents, and service-delegates are based on our previous work that has been done in the Self-Serv project [24].

A supervisor-agent handles the management of the meeting platform. This platform is a common place in which agents meet and interact locally [16]. User-delegates interact with service-broker-agents and resource-broker-agents in the meeting platform. The advertisement of services and resources are sent directly to broker-agents, assuming that the contact addresses of these brokers are known to resource-delegates and service-delegates. Meanwhile, the services that users have decided to trigger in order to satisfy their needs are submitted to the supervisor-agent that assigns their performance to user-delegates. The supervisor-agent has a pool from which user-delegates are created. Besides the user-agent, each user will have a user-delegate in the meeting platform.

Figure 2 summarizes the major types of interactions that occur between the agents and delegates of the agent-based architecture. A part of the numbering (1...5) of the interactions that is used in this figure matches the numbering that is used in figure 1.

4. Service execution planning in hybrid environments

In section 4.1, we briefly discuss the modeling of services using statecharts. Section 4.2 proposes a data model for the description and advertisement of services and resources. Section 4.3 presents a cost model for selection of execution plans of a composite service. Finally, section 4.4 presents an algorithm that is used for building an optimal execution plan of a composite service.

4.1. Modeling composite services with statecharts

A composite service is expressed as an aggregation of other services (either Web services or M-services) using statecharts [13]. Encoding the flow of operation invocations as statecharts is especially attractive for several reasons. First, statecharts possess a formal semantics, which is essential for reasoning about composite service specifications. Next, statecharts are becoming a standard process-modeling language as they have been integrated into Unified Modeling Language (UML). Finally, statecharts offer most of the control-flow constructs found in existing workflow specification languages (branching, concurrent threads, structured loops). More interestingly, it has even been shown in [10] that some useful workflow patterns can be expressed in statecharts, which can not be expressed in the specification languages of many commercial Workflow Management Systems (WfMS). It should be noted that the composition of services using statecharts is based on our previous work [4]. Here, we give a brief overview in order to keep the paper self-contained.

A statechart has *states* and *transitions*. Transitions are labelled by ECA (Event Condition Action) rules. When a transition fires, its action part is executed and its target state is entered. The event, condition, and action parts of a transition are all optional. A transition without an event is said to be *triggerless*.

States can be *basic* or *compound*. In our approach, a basic state corresponds to the execution of a service (e.g., an M-service for attraction search). Compound states contain one or several entire statecharts within them. There are two kinds of compound states: OR and AND states. An OR-state contains a single statechart, while an AND-state contains several statecharts (separated by dashed lines) which are intended to be executed concurrently. Each of these statecharts is called a *concurrent region*. A more comprehensive description of statecharts can be found in [13].

For illustration purposes, figure 3 shows *travel assistant composite service* TAS. It is composed of several connected services: *flight booking* (FB), *accommodation booking* (AB), *attraction search* (AS), and *car rental* (CR). There is an AND-state in TAS in which AS is performed in parallel with AB. In addition, AS and AB are M-services so that user can perform the accommodation booking and attraction searching at her flexible time (e.g., she can do this on the way to the airport using her PDA).

Travel Assistant Service (TAS)

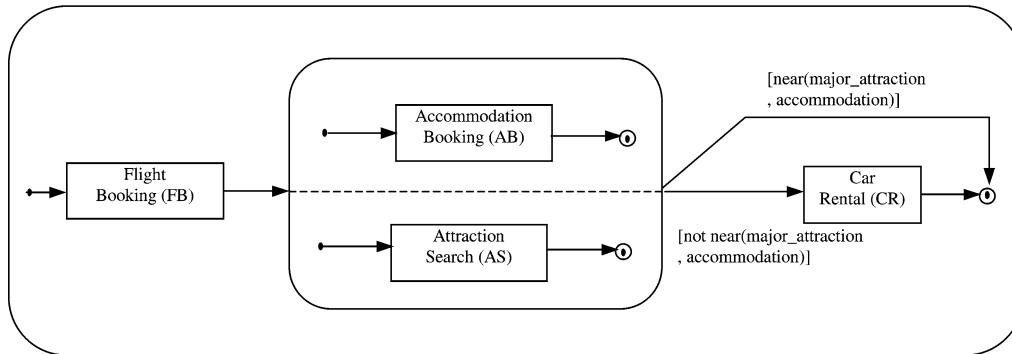


Figure 3. Travel assistance composite service.

4.2. Advertising services and resources

The providers of resources or services need to advertise the characteristics and requirements of the resources or services (figure 1, interactions (1) and (1')). The resource (respectively, service) descriptions are defined by the providers and sent to the resource-broker-agent (resp., service-broker-agent) for registration.

4.2.1. Resource description

Two important features of the resource data model are: (i) it is a semi-structured data model. No specific schema is required so that the system can work naturally in a heterogeneous environment, and (ii) the data model allows providers to express constraints on the services they are willing to serve.

There are two main parts in the description: *attributes* and *constraints*. The Attributes part includes characteristics of a resource, such as location, CPU usage, and free memory. Note that the values of some characteristics change over time (i.e., *dynamic* characteristics). The amount of free memory, free storage space, and CPU usage are samples of characteristics. The values of dynamic characteristics can be obtained via a daemon process running on the resource. The Constraints part includes constraint expressions defined by the resource provider for the allocation of this resource. For example, the resource will not be offered to any process from company A (e.g., it always delays the payments).

Figure 4 shows the description of a workstation. In this example, the workstation has 512 megabytes free memory running SUN Solaris 2.5. The workstation is provided by UNSW and located at hummel.cse.unsw.edu.au. The monetary processing cost charged by this resource is 3 electronic coins² per minute. In reality, a resource provider can publish multiple prices for different processing cycles (figure 5).

² We use *electronic coin* instead of US dollar as the unit of price to make our system globally applicable.

```

Attributes:
  resource-identifier = "UNSW-DB001";
  type                = "Machine";
  description         = "workstation of DB group at UNSW";
  provider-identifier = "UNSW007";
  diskfree            = 10240; //megabytes
  memoryfree         = 512;   //megabytes
  cache              = 8;     //megabytes
  OpSys              = "SOLARIS251";
  location            = "hummel.cse.unsw.edu.au";
  price              = 3;     //electronic coins/minute
Constraints:
  untrusted           = {"doc.it.ac.jp", "esc.soony.net"}

```

Figure 4. Data model describing a workstation.

```

peakttime_price      = 10; //9am-6pm: office hours
off_peakttime_price = 7;  //6pm-9am
holidaytime_price   = 5;  //during holidays and week ends

```

Figure 5. Pricing example.

Further, the workstation will not accept requests from doc.it.ac.jp and esc.soony.net because they are listed in the untrusted clause of the Constraints part.

4.2.2. Service description

In a similar manner to a resource description, service data model has also two main parts: *Attributes* and *Requirements*. The Attributes part includes characteristics of a service, such as location, service provider, input parameters, and output parameters. Although one service can have multiple operations, we consider one operation for clarity reasons. The Requirements part includes execution needs of the service towards the available resources. For example, a service requires at least 32 megabytes of free memory to be executed.

Figure 6 shows a description of an M-service. In the example, the service charges 10 electronic coins each time for the execution. To execute this service, a resource must have more than 20 Kbytes of free disk space and at least 128 Kbytes of free memory. Furthermore, the service can be executed under Palm OS environment.

More specific to a mobile environment are proxies, avg-size-of-input, and avg-size-of-output properties. The communication performance is a main concern. The attributes avg-size-of-input and avg-size-of-output can be helpful in deciding where a service should be executed in order to achieve the best communication performance. For example, assume that service A receives only 20 K bytes as its input, but the output will be 20,000 K bytes. In addition, the output needs to be sent to service B. In this case, it is better to execute A at the site of B, if A can be transferred to other sites and the resource at the site of B satisfies the execution requirements of A. The values of the attributes can be derived from the execution history. Each service stores a history of

```

Attributes:
  service-identifier = "axac267";
  type               = "M-service"; //Web or M-service
  isMobile           = "yes"; //whether the service can
                        //move to other places
  description        = "accommodation booking service";
  provider-identifier = "XAC";
  location           = "cts.xac.com.ae";
  input-parameters  = {Int NumofPersons, Int NumofDays,
                        String ContactName};
  output-parameters = {XMLDoc accommodationDetails};
  avg-size-of-input  = 2; //Kbytes
  avg-size-of-output = 20; //Kbytes
  price              = 10; //e-coins per invocation
  proxies            = {"UNSW007", "CR001", "ATP117"};
Requirements:
  diskfree           > 20; //Kbytes
  memoryfree         >= 128; //Kbytes
  OpSys              = "Palm OS"

```

Figure 6. Data model describing a service.

service execution in a local log, along with the associated values of the relevant attributes (e.g., size of input/output parameters).

The attribute `proxies` is used to avoid service code transmission by keeping copies of the service in the sites of some *companion* resources. Consequently, the transmission time is avoided because the service provider does not need to send the service if it is executed using one of these companion resources. The companion resources can be manually defined by service provider (e.g., he may prefer to use the resource UNSW007), or derived from the execution history of the service (e.g., service provider may choose those resources used frequently as companion resources of the service.).

4.3. Cost model

The components of a composite service can be executed using several resources (interactions in figure 1). Therefore, there can be multiple *execution plans* for a composite service. By execution plan, we mean those resources which can be used to execute a composite service.

Assume a composite service S has m component services, $S_c = \{s_1, s_2, \dots, s_m\}$, an execution plan p of S is defined as follows:

Definition 1 (Execution plan). $p = \{\langle s_1, r_1 \rangle, \langle s_2, r_2 \rangle, \dots, \langle s_m, r_m \rangle\}$ is an execution plan of composite service S if:

- $\bigcup_{i=1}^m s_i = S_c$, and
- for each 2-tuple $\langle s_i, r_i \rangle$ ($i \in [1, m]$) in p , the service s_i is executed on resource r_i .

In fact, an execution plan indicates which resource is used for each component. Note that the number of resources is not necessarily equal to the number of component services because some component services could share one resource in order to reduce the communication costs.

Each component service is associated with a *scoring function*. A scoring function interprets a *selection policy* which specifies preferences over resources of a component service. It is a *multi-criteria utility function* [25] which has the following form:

$$U(r) = \sum_{i \in SC} w_i \cdot Score_i(r), \quad (1)$$

where:

- $Score_i(r)$ is a criteria scoring function, which, given a value of a criterion i of a resource r , returns a score (e.g., a positive integer value). SC is the set of selection criteria.
- w_i is the weight assigned to the criterion i .

The scoring service computes the weighted sum of criteria scores using the weight property of each selection criterion. It selects the resource of a component service that produces the higher overall score according to the multi-criteria utility function. Several criteria (SC set), such as *price*, *execution time*, *availability*, *reliability*, and *reputation*, can be used in the function. Currently, in our approach, we consider only two criteria: price and reliability.

4.3.1. Price

Given a resource r of a component service s , the price $P(r)$ is

$$P(r) = p(r), \quad (2)$$

where p is charged by resource r to service s for its execution. The value of the price is obtained from the resource advertisement (section 4.2).

4.3.2. Reliability

Given a resource r of s , the reliability of the resource $R(r)$ is

$$R(r) = N_c(r)/K, \quad (3)$$

where $N_c(r)$ is the number of executions successfully performed by r for the last K requests. Here K is a constant set by an administrator.

For instance, the scoring function associated with the criterion price is $Score(P(r)) = 1/P(r)$, i.e., the higher the price, the lower the score. It should be noted that the method of estimating the value of a criteria is not unique, neither is the set of criteria.

Another important criterion proposed in our approach is *location of computing resources*. The location criterion aims at gathering the maximum number of components of a composite service to be executed in the same site (i.e., using the same computing

resource). As a consequence, remote interactions and communications between component services can be avoided. The selection of resource for a component depends on the location (i.e., computing resource) of its predecessor. We will show how the location criterion is used in section 4.4.

4.4. Service execution planning

To execute a composite service, an execution plan should be selected (i.e., which computing resource should be used for each component of the composite service). Building an execution plan consists of several phases.

Phase 1: Matching resources. The purpose of this phase is to search for the resources on which the component services could be executed. In the meeting platform, user-delegates trigger the matching process between services and resources.

If all the requirements of a component service s_i are evaluated as `true` using the attributes of a resource r_i , we say r_i is a *matched resource* of s_i . A reference to a non-existent attribute evaluates to the constant `undefined` which is treated as `false`. The results of the matching phase of S are:

$$R_i = \{r_{i1}, r_{i2}, \dots, r_{ini}\}, \quad (4)$$

$$M = \{\langle s_1, R_1 \rangle, \langle s_2, R_2 \rangle, \dots, \langle s_m, R_m \rangle\}, \quad (5)$$

where R_i is the set of matched resources of service $s_i \in S_c$. M is the matching results of S .

For example, in figure 3, the matching results of TAS could be: $\{\langle \text{FB}, \{r_1, r_2, r_3\} \rangle, \langle \text{AB}, \{r_1, r_7\} \rangle, \langle \text{AS}, \{r_7, r_9, r_{10}\} \rangle, \langle \text{CR}, \{r_4, r_6, r_7\} \rangle$. We can see that resources r_1, r_2 and r_3 can be used to execute service `flight booking`. The resource r_1 can also be used to execute service `accommodation booking`. The descriptions of resources $r_1, r_2, r_3, r_4, r_6, r_7, r_9$, and r_{10} are not given here for reasons of brevity.

Phase 2: Pruning phase. For services which are in different concurrent areas of an AND state in the statecharts of S , if a resource r is shared by several component services, each component service belongs to different concurrent areas of the AND state, a *rematch procedure* must be performed to ensure that these component services can be executed in r concurrently. For example, in figure 3, AB and AS require at least 150 Kbytes and 120 Kbytes of free memory to carry out their executions, respectively. Resource r_7 has 260 Kbytes of free memory. Intuitively, r_7 meets the requirements of both AB and AS separately. However, r_7 does not meet the requirements when AB and AS are executed at the same time because they need 270 Kbytes totally. As a result, the resource r_7 is removed from the sets of the matched resource of AB and AS.

Since each component service s_i ($s_i \in S_c$) has a set of matched resources (R_i), the association of a component service with a specific resource has to be completed in order to build an execution plan for composite service S .

Step 1: Initially, the work starts with the first component service s_i ($i = 1$) of the composite service S . In this step, a best resource will be selected from R_i using the

```

Algorithm
∀i, i = 2, ..., m
for each ⟨si, Ri⟩
if (ri-1 ∈ Ri)
then begin
    establish ⟨si, ri-1⟩ //ri-1 is selected to execute si
end
else begin
    go to Step 1 //using utility function to select
    //a resource for si
end

```

Figure 7. Algorithm for resource selection.

cost model we developed in (1). The score of a resource r ($r \in R_i$) is computed using the following formula:

$$U(r) = w_p \cdot Score_p(r) + w_r \cdot Score_r(r). \quad (6)$$

Here w_p and w_r are weights for price and reliability, respectively. The scoring functions (i.e., $Score_p(r)$ and $Score_r(r)$) are provided by the composite service provider. However, end users can customize the weights of the selection criteria in order to find a desired resource for the component service. For example, if the price is the most important factor to a customer, she can set w_p to 1 and w_r to 0. For each resource, a score is computed using above utility function and the resource with the maximal value of $U(r)$ is selected. If there is more than one resource which have the same maximal value of $U(r)$, then a resource will be chosen randomly from them. Suppose resource r_i is finally selected to execute service s_i , the plan for executing service s_i is represented as $\langle s_i, r_i \rangle$.

Step 2: After the preparation of the first component service s_i ($i = 1$) is finished, the resources should be selected for the remaining component services s_i ($i = 2, \dots, m$). The location criterion is considered at this stage. Note that the location criterion is privileged to the other two criteria because of the reasons previously discussed. Obviously, the resource that is going to be assigned to a component service s_i ($i = 2, \dots, m$) depends on the location of the resource that is assigned to its predecessor s_{i-1} . Figure 7 illustrates the algorithm that is adopted for resource selection ($\langle s_{i-1}, r_{i-1} \rangle$ is already established). Finally, the execution plan p is built. We say that the built execution plan is an *optimized* execution plan of composite service S .

After the optimized execution plan of S is built, the user-delegate executes the composite service. In fact, the user-delegate is in charge of initiating, controlling, and monitoring the execution of S . The knowledge required at runtime (i.e., control flow routing policies) is statically extracted from the statechart of the composite service S . It is important to note that the execution sequence of a composite service is implicitly specified in the statecharts of the service.

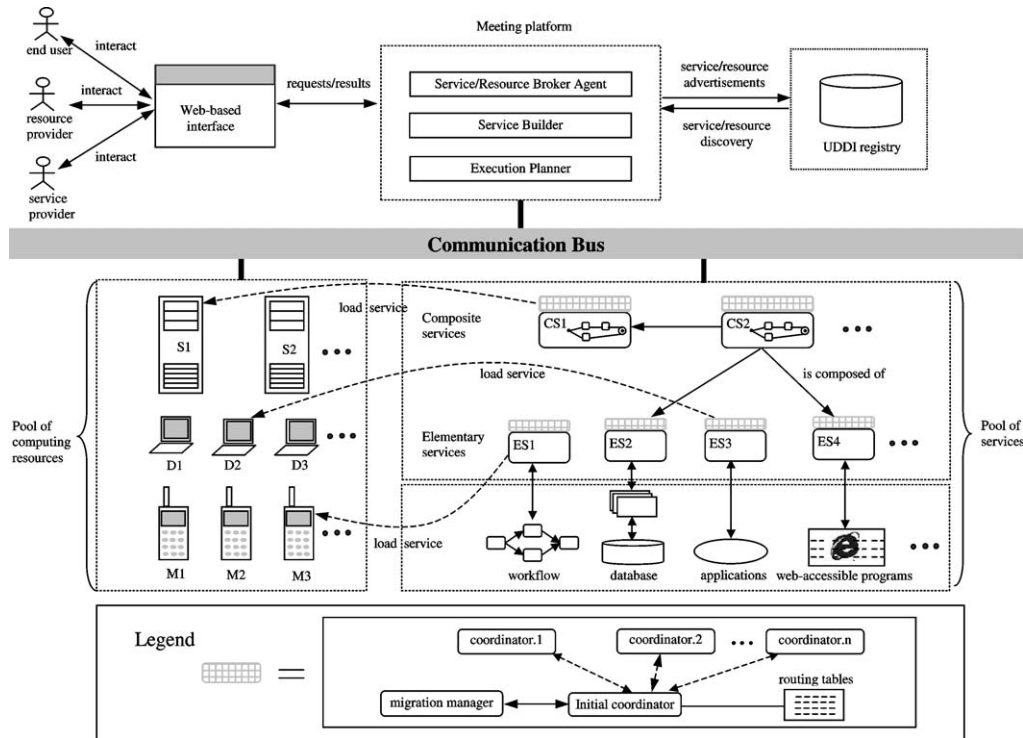


Figure 8. Architecture of the prototype.

5. Implementation status

In this section, we overview the status of the prototype implementation of the service composition approach. Until now, the implementation has shown that the ideas behind the approach fit together, are consistent with one another, and are realizable using existing technologies.

The prototype architecture (figure 8) consists of an *interface*, a *meeting platform*, and a *pool of services and resources*. The *meeting platform* consists of three modules: *service/resource broker agent*, *service builder*, and *execution planner*. All these components have been implemented in Java. Services communicate through XML documents. Aglets are used to implement mobile agents for service invocation at resource sites. IBM's Aglets Software Development Kit v2 is used for implementing Aglets. Sun's Java 2 Platform Micro Edition (J2ME) Wireless Toolkit 1.0.4-01 is used for implementing M-services. Section 5.1 briefly describes the pre-built agents of the meeting platform. Section 5.2 presents the service composition environment.

5.1. Pre-built agents

For any service wishing to participate in our platform, the administrator of the service needs to download and install a set of pre-built agents, namely *user-delegate* and *service-*

delegate (i.e., service-provider-agent). The service delegate is composed of a *service wrapper* and a *service migrator*. The functionalities of the user-delegate are realized by a class called *userDelegate*. The only infrastructure required to install and configure these agents, are Java, Tahiti (a tiny Aglet server program), and the XML parser.

The class *userDelegate* provides methods for receiving, processing, generating and sending control-flow notifications, service invocation, and service completion messages. The concept of service wrapper is mapped into an abstract class called *Wrapper*; which defines (among others) methods for (i) invoking an operation of the service (method `start_service`), and (ii) collecting the outputs of a service instance (method `get_service_result`).

When the wrapper receives notifications from a user-delegate, the notifying message is passed to *Migrator* if the service will be executed in another resource host. The *Migrator*, that implements service migrator, is responsible for: (i) uploading the service to the resource host; and (ii) dispatching a mobile agent to the host for service invocation and result collection.

We have adopted Aglet [1] for implementing mobile agents which invoke services in the resource hosts and return the invocation results. *ServiceInvoker* is an Aglet which needs to be downloaded together with the *Migrator*. It is a static Aglet. When a service needs to move to a resource host, `serviceInvoker` creates a slave called `serviceInvokerSlave`, and passes the destination (e.g., resource host) to the `serviceInvokerSlave`. *ServiceInvokerSlave* is the labor Aglet that actually goes to the resource site. Upon arrival at the resource site, `doJob` method of the `serviceInvokerSlave` is called, which performs the real work that we assign to the slave (i.e., invoke the service). When the `serviceInvokerSlave` returns to the service site, `callBack` method of the master Aglet `serviceInvoker` is activated. The results are passed as an argument of the `callBack` method. The `serviceInvoker` Aglet then extracts and passes the results to the *Migrator*. The *Migrator* in turn sends the invocation results to the *Wrapper*.

5.2. Service definition and execution environment

The service composition environment consists of a set of integrated tools that allow service-delegates and user-delegates to create and execute services. It is composed of the following component tools: service/resource broker agent, service builder, and execution planner.

The service/resource broker agent facilitates the advertisement and location of services and resources. In the implementation, the Universal Description, Discovery and Integration (UDDI) is used as service/resource repository. We define an XML schema for resource description (section 4.2). Each resource is represented as an XML document. For each resource, a tModel of type `resourceSpec` is created. The tModel includes a tModel key, a name (i.e., resource name), an optional description, and a URL that points to the location of the resource description document. The Web Service Description Language (WSDL) is used to specify Web services. Since WSDL focuses on how to invoke

```

<tModel tModelKey="uuid:9856f21e-badd-492b-21rt408f6645">
  <name>serviceType</name>
  <descrip. lang="en">Specification service type</descrip.>
  <overviewDoc>
    <description lang="en">service type</description>
    <overviewURL>http://host:80/serviceType.xml</overviewURL>
  </overviewDoc>
</tModel>

```

Figure 9. serviceType tModel.

a Web service, some of the attributes in our proposed service description model are not supported by WSDL (e.g., service type: Web service or M-service). Such attributes are specified as tModels. Each tModel represents the specification of one attribute. Figure 9 shows the example of serviceType tModel. The specification of the service type is described in an XML document `http://localhost:80/serviceType.xml`. The keys of these tModels are included into the categoryBag of the tModel of a Web service.

The service builder assists providers through their service delegates in defining new services and editing existing ones. A service definition is edited through a visual interface (figure 10), and translated into an XML document for subsequent analysis and processing by the service deployer. The service builder offers an editor for describing a statechart diagram of a composite service. It also provides means to describe the properties of states (e.g., state ID, state name, component service operation) and transitions (e.g., EGA rules).

Finally, execution planner is the module that plans the execution of a composite service using the approach presented in section 4.4. A search facility is offered to locate services and resources using the service/resource broker agent.

6. Related work

There exist several research projects that aim at studying how mobile devices can change our way of doing business and performing operations [6,9,12,20,21,26]. At HP, Milojevic et al. have worked on delivering Internet services to mobile users [21]. This work was conducted under the project Ψ for Pervasive Services Infrastructure (PSI). Ψ 's vision is "any service to any client". The project investigated how offloading parts of applications to mid-point servers can enable and enhance service execution on a resource-constrained device. In our research, we are interested in the same issues as Ψ . Furthermore, we are interested in putting forward "any service to any client and for any device". Users should not be concerned about the type of services that will be involved (Web services or M-services) nor about the type of computing resources (fixed or mobile) on which these services will be executed.

Within the context of Ψ , Messer et al. have suggested the creation of an ad-hoc distributed platform to transparently offload portions of a service from a resource-constrained device to a nearby server [20]. In fact, if a device becomes resource con-

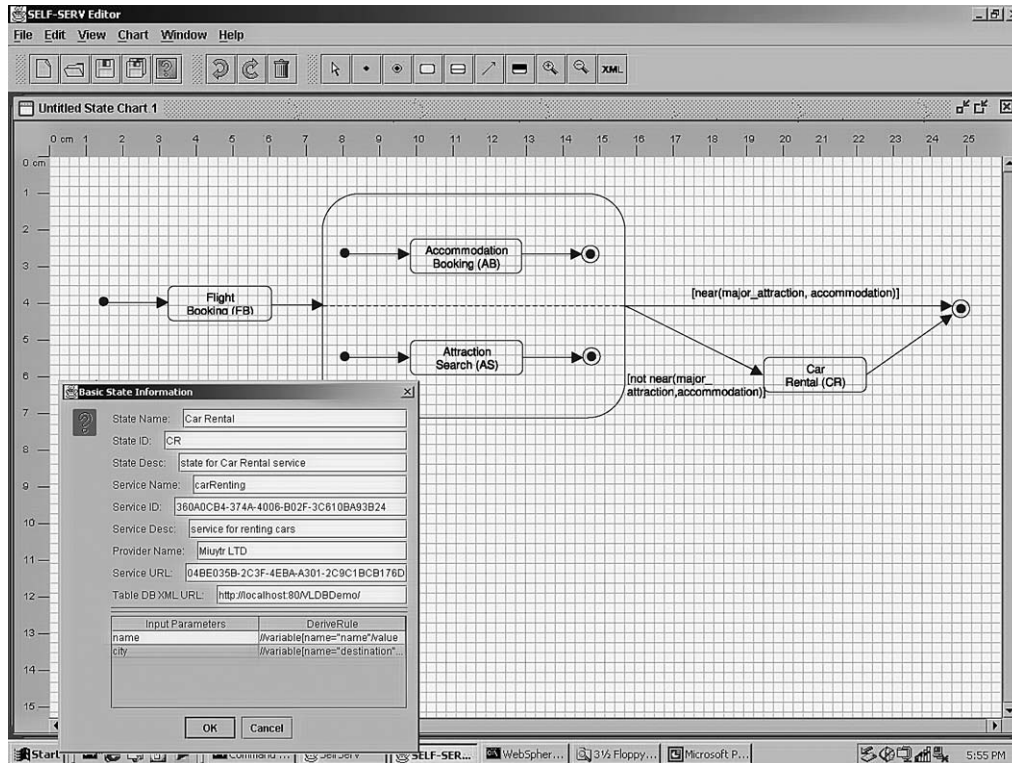


Figure 10. Composing services using the platform.

strained at run time and believes it can beneficially use nearby resources, it automatically and transparently offloads part of the service to them. The suggested platform can dynamically decide how much of the surrounding resources to use. The platform philosophy presents similarities with the service composition approach. In this approach all the services are offloaded from their original sites to the computing resource sites that have the capacity to handle their execution. We offload the whole services and not portions of them based on the commitments that resource sites will provide. Commitments are an important element of the approach operation.

Chakraborty et al. have claimed that there are situations where the information is not available from a single service provider but can be obtained by combining information from multiple service providers [8]. Due to the increasing number of users who seek information while on the move and due to the complexity of coordinating multiple services from mobile units, an agent-based middleware architecture has been suggested in [8]. The central concept of the design of this architecture is a broker-agent that resides in a wired infrastructure and handles queries from mobile units. In the service composition approach, the meeting platform corresponds to the wired infrastructure and has been implemented on top of a fixed platform. Because several agents and delegates in the architecture reside in the meeting platform, disconnections in the wireless chan-

nel have little impact on the operation of the approach. For instance, a user-delegate allows a user to disconnect himself from the network after triggering a service, and to reconnect again upon receiving notifications from his user-delegate. The meeting platform has also allowed user-delegates to undertake their operations without being concerned with the scarcity of the computing resources that feature certain mobile devices.

Chakraborty et al. have also introduced a reactive service composition architecture for pervasive computing environments [7]. The architecture consists of five layers: network, service discovery, service composition, service execution, and application. While reviewing their 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 have introduced in the service composition algorithm. With that criterion, the cross-network traffic between the resources can be reduced. This avoids extra data transfer between distant resources.

7. Conclusion and future work

In this paper, we presented our approach on provisioning Web services in hybrid environments. A hybrid environment consists of various types of computing resources, fixed ones such as desktops and mobile ones such as cell-phones. The backbone of this framework is an agent-based architecture that integrates several agents such as user, provider, service, and resource.

Our approach is one step towards provisioning Web services independently of the resources that are needed in the performance of these services. This approach integrates three selection criteria (execution cost, reliability, and resource location) to identify which resources should be assigned to which services. The location criterion has been privileged over the other two criteria because of the advantages that it provides when resource-constrained devices intervene. This criterion enables the execution of the maximum number of Web services on the same resources assuming that these resources have the capacity of meeting the execution requirements of the Web services. Using the location criterion, remote interactions and communications between component services can be avoided.

Our ongoing work includes the assessment of the performance and scalability of the proposed service execution planning approach. We also plan to examine the support for exception handling during the execution of composite services in hybrid and dynamic environments. It may happen that a resource change (e.g., disconnection due to discharged battery, change of location) affects the execution of the whole composite service. We plan to explore agent support for performing dynamic re-planning of service execution.

Acknowledgements

The third author's work was in part supported by an Australian Research Council (ARC) Discovery grant #DP0211207. The authors would like to thank the guest editors and reviewers for their comments and suggestions of improvements.

References

- [1] Aglet, http://www.trl.ibm.com/aglets/index_e.html, visited June 2002.
- [2] P. Bellavista, A. Corradi and C. Stefanelli, Mobile agent middleware for mobile computing, *IEEE Computer* 34(3) (2001).
- [3] B. Benatallah and F. Casati (eds.), *Distributed and Parallel Databases 12(2-3)*, Special Issue on Web Services (2002).
- [4] B. Benatallah, M. Dumas, Q.Z. Sheng and A. Ngu, Declarative composition and peer-to-peer provisioning of dynamic web services, in: *Proceedings of the 18th International Conference on Data Engineering (ICDE'2002)*, IEEE Computer Society, San Jose, USA (2002).
- [5] B. Benatallah, Q.Z. Sheng and M. Dumas, The self-serv environment for Web services composition, *IEEE Internet Computing* 7(1) (January/February 2003).
- [6] G. Caire, N. Lhuillier and G. Rimassa, A communication protocol for agents on handheld devices, in: *Proceedings of the 1st International Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices held in conjunction with the 1st International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'2002)*, Bologna, Italy (2002).
- [7] D. Chakraborty, F. Perich, A. Joshi, T. Finin and Y. Yesha, A reactive service composition architecture for pervasive computing environments, in: *Proceedings of the 7th Personal Wireless Communications Conference (PWC'2002)*, Singapore (2002).
- [8] D. Chakraborty, F. Perich, A. Joshi, T. Finin and Y. Yesha, Middleware for mobile information access, in: *Proceedings of the 5th International Workshop on Mobility in Databases and Distributed Systems (MDDS'2002), held in conjunction with DEXA'2002*, Aix-en-Provence, France (2002).
- [9] I. Chisalita and 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).
- [10] M. Dumas and A. ter Hofstede, UML activity diagrams as a workflow specification language, in: *Proceedings of the International Conference on the Unified Modeling Language (UML'2001)*, Toronto, Canada (2001).
- [11] I. Elsen, F. Hartung, U. Horn, M. Kampmann and L. Peters, Streaming technology in 3G mobile communication systems, *IEEE Computer* 34(9) (2001).
- [12] P. Fenkam, E. Kirda, S. Dustdar, H. Gall and 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, PA, USA (2002).
- [13] D. Harel and A. Naamad, The STATEMATE semantics of statecharts, *ACM Transactions on Software Engineering and Methodology* 5(4) (1996).
- [14] A.C. Huang, B.C. Ling, S. Ponnkanti and A. Fox, Pervasive computing: What is it good for? in: *Proceedings of the Workshop on Mobile Data Management (MobiDE'99) in conjunction with ACM MobiCom'99*, Seattle (1999).
- [15] N. Jennings, K. Sycara and M. Wooldridge, A roadmap of agent research and development, *Autonomous Agents and Multi-Agent Systems* 1(1) (1998).
- [16] Z. Maamar, E. Dorion and C. Daigle, Towards virtual marketplaces for e-commerce, *Communications of the ACM* 44(12) (2001).

- [17] Z. Maamar, W. Mansoor and Q.H. Mahmoud, Software agents to support mobile services, in: *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'2002)*, Bologna, Italy (2002).
- [18] Z. Maamar, W.J. van den Heuvel and W. Mansoor (eds.), *Workshop on M-Services/Approaches – Concepts – Tools/, workshop held in conjunction with The 13th International Symposium on Methodologies for Intelligent Systems*, Lyon, France (2002), <http://www.elcomag.com/MServices2002Workshop/>.
- [19] P. Maes, Agents that reduce work and information overload, *Communication of the ACM* 37(7) (1994).
- [20] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T.J. Giuli and X. Gu, Towards a distributed platform for resource-constrained devices, in: *Proceedings of the IEEE 22nd International Conference on Distributed Computing Systems (ICDCS'2002)*, Vienna, Austria (2002).
- [21] D. Milojicic, A. Messer, P. Bernadat, I. Greenberg, G. Fu, O. Spinczyk, D. Beuche and W. Schroder-Preikschart, Ψ 's – Pervasive Services Infrastructure, Technical report HPL-2001-87, HHP Laboratories, Palo Alto, CA, USA (2001).
- [22] I mode FAQ, <http://imodelinks.com/desktop/faq.html/>, visited March 2002.
- [23] T.S. Raghu, R. Ramesh and A.B. Whinston, Next steps for mobile entertainment portals, *IEEE Computer* 35(5) (2002).
- [24] Q.Z. Sheng, B. Benatallah, M. Dumas and 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).
- [25] M. Stolze and M. Stoebel, Utility-based decision tree optimization: A framework for adaptive interviewing, in: *Proceedings of the 8th International Conference on User Modeling (UM'2001)*, Sonthofen, Germany (2001).
- [26] The Ninja project, <http://ninja.cs.berkeley.edu>, visited September 2001.
- [27] S.J. Vaughan-Nicholas, Web services: Beyond the hype, *IEEE Computer* 35(2) (2002).
- [28] R. Vingralek, Supporting e-commerce in wireless networks, Technical report, INTERTRUST STAR*LAB Technical Report STR-TR-01-07, InterTrust Technologies Corporation, CA, USA (2001).
- [29] D. Wodtke and G. Weikum, A formal foundation for distributed workflow execution based on state charts, in: *Proceedings of the 6th International Conference on Database Theory*, Delphi, Greece (1997).