Using Context Information for Service Discovery and Composition

S. Kouadri Mostéfaoui, Amine Tafat-Bouzid and B. Hirsbrunner

Department of Computer Science
University of Fribourg
Chemin Du Musée 03, Pérolles
CH-1700, Fribourg, Switzerland

Abstract. In principle, with Web services application creation is simply a matter of discovering, selecting the right services and composing them into a solution [1]. Existing approaches for service discovery and composition, typically facilitate orchestration only, while neglecting information about the context of users and services as well. However while computing today is becoming increasingly pervasive there is an unaddressed need of more location and context-aware support. In this paper we present a generic framework that combines service-oriented and context-aware computing in order to provide users with more tailored composite services.

Keywords. Web services, service discovery, service composition, context.

1 Introduction

Web services have become a new wave of internet technology development. It is a new solution for dynamic business interactions over the internet. In recent years, most major industry players have ventured into this area, staking a claim in their future [2]. At present, the technological infrastructure for Web Services is structured around three major standards: WSDL (Web Service Definition Language), UDDI (Universal Description, Discovery, and Integration), and SOAP (Simple Object Access Protocol) [3]. These technologies aim at supporting the definition of Web services, their advertisement to the community of potential users, and finally their binding for triggering purposes. A more challenging problem is to compose services dynamically, on demand [4]. In particular, when a functionality that cannot be realized by the existing services is required, the existing services can be combined together to fulfill the request.

Recently several research projects on the field of service composition have been proposed, for example, the dynamic service composition called software hotswapping [22] at the Carleton University [23], the eFlow [24] from HP laboratories, and the Ninja [25] service composition at the University of California Berkeley [21]. These approaches typically facilitate orchestration only, while neglecting information about the context of users and services as well. However while computing today is becoming increasingly pervasive there is an unaddressed need of more location and context-aware support. In this paper we describe our initial research work on CB-SeC a system that allows services discovery and composition taking benefits from the available context information.

The rest of this paper is as follows. Section 2 overviews the concepts of Web vs. mobile services and context respectively. In section 3 the main components of the CB-SeC framework are introduced. Section 4 and 5 describe the context layer and the middleware services layer respectively. A first application scenario is presented in section 6. Finally Section 7 draws the conclusions and presents our future work.
2 Background

2.1 Web services

W3C has defined a Web service as a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols [2] [6].

2.2 Mobile Services

Maamar et al. introduce the concept of M-services (M for Mobile) as a specific type of Web services [7] [8]. Two definitions are suggested. The weak definition is to remotely trigger a Web service from a mobile device for execution. 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; and finally (iv) runnable on mobile devices.

2.3 Context

Composed of con (with) and text, context refers to the meaning that can be inferred from the adjacent text. Dey considers context as any information about the circumstances, objects, or conditions by which a user is surrounded that is considered relevant to the interactions between the user and the environment [9]. Many researchers have attempted to define context by enumerating examples. Schilit et al. divide context into three categories [10], 1) computing context such as network connectivity, communication costs, communication bandwidth, and nearby resources such as printers, displays, and workstations; 2) user context such as user’s profile, location, people nearby, and even sometimes current social situation; and 3) physical context such as lighting, noise levels, traffic conditions, and temperature.

3 Overview of the Architecture

The architecture of the CB-SeC framework is depicted in Fig. 1. CB-SeC is built around a classical layered model: physical, context, service and application layers.

The Physical layer: represents the resources belonging to the environment. Resources describe all the entities that may be involved in the execution process of an application; so they describe physical devices, software components and sensors. Due to the dynamic changes in the availability of physical devices, network bandwidth, connectivity and user location, the system has to classify those resources monitoring constantly their changes. In such a way the upper layers have always a consistent description of the physical dimension [11].

The Context layer: this layer is responsible for gathering and processing contextual information. It is composed of two modules: the Context Gathering Engine, and the Context Data Base. The Context Gathering Engine consists of a Multi-Agent System in which each agent is responsible for gathering and processing one type of context information, and for providing this information to the Context Data Base. The contextual information is acquired either through the sensors, or through the use of special messaging between the agents in the system.

1 CB-SeC stands from Context-Based Service Composition.
The **Middleware Services layer**: this layer is the core part of our framework, it is responsible for carrying out the process of discovery, composition and execution of services, it is in turn composed of four main modules, the Context-Based Service Discovery module, the Context-Based Service Composition module, the Service Execution module, and the Cache Engine. These modules are tightly coupled with each other due to their inter-dependencies; each of them will be described in details when in section 5.

The **End-User Application layer**: is a generalization of the view in a traditional Model Viewer Controller (MVC) [13]; it works as a presentation module embodying some application dependent functionalities like visualisation that produces, input queries. Input queries are represented as a composition of basic services using UCM [15]. The End-User Application layer encompasses also different GUI facilities with whom users can customize their applications. In order to provide more personalized composite services [11].

### 4 The Context Layer

The operating of the context-awareness approach in CB-SeC consists of two steps: context acquisition, and context representation. This is similar to what Chen et al. denote in [12] by sensing the contextual information, and reasoning about contextual knowledge. For each of these steps we have associated a computational module, respectively the Context Gathering Engine and the Context Data Base, each of them will be described in turn after identifying the context types we used in CB-SeC.

#### 4.1 Using Context in Service Discovery and Composition

From our perspective three different types of sources can contribute to the constitution of a context. These sources are: services, users, or both users and services. In this paper, both of the two types are used. The user context includes user’ information (e.g. user role, identity, age, location, preferences, permission profile...etc). The service context includes all service related information (e.g. number of instances allowed vs. number of currently running instances, execution requirements in terms of computing resources...etc).

A user-centric context promotes applications that 1) move with users, 2) adapt according to changes in the available resources, and 3) provide configuration mechanisms according to users’ personal preferences [18]. And, a service-centric context promotes ap-
plications that 1) allow service adaptability and scalability, 2) enable quality-of-service selection, and 3) support on-the-fly service composition.

4.2 Abstract Representation of Context Information

To proliferate context information in a timely due manner and general format, a representation of context information based on the resource description framework (RDF) [19], called the CSCP (Comprehensive Structured Context Profiles) is used. First introduced by A. Held and al. in [20], this representation expresses information by means of session profiles. A session profile describes all relevant context information of the session, in our case a session corresponds to the request taken by the Broker Agent. The CSCP is an RDF based meta language. As a descendant of RDF, CSCP inherits the interchangeability, decomposability and extensibility of RDF. CSCP interchangeability is based on the XML serialization syntax of RDF [19]. Furthermore, CSCP provides features to attach conditions and priorities to attributes. This extends the means to express user preferences. By assigning conditions to user preference attributes, by means of multiple conditional RDF statements about an attribute, If-Then-Else-If expressions may be formulated.

4.3 The Context Gathering Engine

The Context Gathering Engine (CGE) is responsible for gathering, processing and interpreting the users contextual information. It consists of a Multi-Agent System (MAS), in which each agent is associated with one type of contextual information. The agents are responsible to gather the data provided by the software and hardware sensors, to interpret the gathered raw sensor data into meaningful information and finally to extract the specifications such as time, location...etc. This does not mean that all the contextual data must be transferred via the wireless link. It may rather contain references to external resources, such as the device defaults that can be retrieved from the device vendors web site [11].

4.4 The Context Data Base

The Context Database (CDB) is acting as a bridge repository; it receives the specifications of the contextual information from the agents of the CGE. These specifications are translated into XML documents for subsequent use by the Context-Based Service Discovery Module of the above layer. An important part of the context database is the classification of the context information into context types, e.g. both the GPS and the network-based locator provide location information. The specification of context types is achieved using an XML template that defines the kind of information this type offers. Using this approach, the Middleware Services Layer can retrieve the specific contextual data in a way that is decoupled from the service used for acquiring the data. This method of hiding the actual mechanism for retrieving contextual information, allows the CB-SeC framework to coordinate the access to context for different applications [14].

5 The Middleware Services Layer

The Middleware Services Layer is responsible for the process of carrying out the discovery, composition and execution of services. It is composed of four modules: the Context-Based Service Discovery Module, the Context-Based Service Composition Module, the Service Execution Module, and the Cache Engine. Each of these modules will be described in turn after introducing how are services modelled in CB-SeC.
5.1 Service Description

There are three main parts in a service description: 1) the Attributes, 2) the Capsule, and 3) the Constraints and Requirements. Another optional part called the Context of Interest Function is shipped with the service description and is used in order to select the best service offers when more than one service is returned by the discovery process.

Attributes*

service-identifier = "pai-accl21";
num instances = 3; number of allowed instances
type = "W-service"; //Web or M-service
isMobile = "No"; //whether the service can move to other places
description = "accommodation booking service";
provider-identifier = "PAI";
input-parameters = {Int Num of Persons, Int Num of Days, String Contact Name};
output-parameters = {XML Doc accommodation Details};
price = 5; //e-coins per invocation

Capsule*

location = "pai-acc.diuf.com.ch";
protocol = https;
port = 80;

Constraints & Requirements*

diskfree > 20; //Kbytes
memoryfree >= 128; //Kbytes
OpSys = "Palm OS, Linux"

Context of Interest Function //this field is optional

CoIF = ping iiufps31.unifr.ch

Fig. 2. Example of a service description.

Attributes*: include the characteristics of a service: i.e. information regarding the operations that can be invoked in a service and their respective input and output parameters. Although one service can have multiple operations, in our example we consider one operation for clarity reasons.

Capsule*: gives information on where the service is located, how it can be accessed, the invocation protocol(s), the types(s) and the port(s).

Constraints and Requirements*: this part wraps all the conditions of use of the service. It is necessary when different providers offer the same service, with different constraints regarding the input and/or output parameters. The Requirements include execution needs of the service towards the available resources. For example, a service requires at least 16 megabytes of free memory to be executed.

Context of Interest Function: this function represents the sensitivity of the service to the context. Unlike the previous three parts, this part is optional and its value is not known in advance, it is calculated in run time if the interface matching process returns back more than one service, in order to help with better service selection. Example of such a function may be the load attribute that can determine which printer is less loaded, i.e. which printer service is better in the current context.

Fig. 2 shows a description of an accommodation booking Web service. In this example the service accepts three input parameters and returns the accommodation details as an XML.
document. The service charges 5 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 or Linux environments. Finally the Context of interest Function (CoIF) identifies the sensitivity of the service to context-parameters, in our example and for simplicity reasons we take a simple IP check function that permits to check weather the provider of a given service is on-line or off-line, in the time the request is generated, Fig. 3.

5.2 The Context-Based Service Discovery Module

The approach adopted in this paper, involves mobile devices that not only consume web services, but also publish their data through web services. Thus an important component of the Context-Based Service Discovery module (CSD) is the Brokering Agent, (BA) that functions is to match and recommend appropriate services among a possibly large set of service instances, taking benefits from context-information.

When a client requests a service it may be necessary to compose a complex service out of the registered basic services. The current implementation of our architecture uses UCMs social laws (UCM stands from Ubiquitous Coordination Model) to decompose complex service requests into basic services, and to determine the process model of execution. The UCM model is an instantiation of the unified coordination model XCM presented in [15]. It is viewed as an organizational domain composed by autonomous entities. An entity is defined by its structure obtained by a recursive composition of entities. Inside this domain, entities interact with each other through communication endpoints that define a set of actions; social-rules are defined to restrict the composition of basic communication actions. More details about UCM are available in [15]. Clients request services in a context describing characteristics, and time of the requested service [21], furthermore context contains information about the client and its preferences. The BA’s first job is to extract the contextual information form the Context-Data Base, then using these information it tries to discover the basic services against the ones registered on its vicinity (using the location attribute), it first matches the users query as well as the users device capabilities (using the service context) into the nearby service registry and progressively increases its search radius [16] to discover all the different services necessary for the composition. The BA has to figure out

![Fig. 3. Selection of the best service after evaluation of the CoIF.](image-url)
other-end requirement, i.e. device capabilities anticipated by service implementations. It can get such information from the constraints field shipped with the services. The previous step produces a set of candidate services, the BA is then responsible for iterating through this set, and for picking the best offers according to the current context and the constraints set by the service providers. In other words services are incrementally filtered and ranked according to the evaluation of the context of interest functions, to ensure that clients are given the best service instances.

5.3 The Context-Based Service Composition Module

This module is responsible for carrying out the process of managing the discovery of services to yield a composite service. It further refines the set of selected services using the remaining context parameters that are relevant for the composition rather than for the discovery. Example of such a parameter can be the user preferences that are applicable to the overall composite service e.g. a person planning for her summer vacation wont spend more than 2000CHF for the whole vacation composite service i.e. (hotel service, car rental service, attractions, plane tickets ...etc), another example may be the permission profile set by the user (e.g. when in class between 8h00 and 12h00 I do not want to receive SMS messages). The composite service that satisfies all the user preferences is chosen and the corresponding capsules (i.e. information about the services, addresses, ports, and invocation protocols) are sent to the Service Execution Module. In the mean time a copy of the process model together with its context is kept in cache.

5.4 The Service Execution Module

This module is responsible for carrying out the execution of the composite service. Prior to this the Context-Based Service Composition module provides a feasible order in which these services can be executed. Mainly two types of execution exist according to the nature of the service, which can be either a Web service (W-service) or a mobile service (M-service). For M-services execution is always local to the device of execution since the service itself has been transported to the device. However for W-services the execution can be remote or local. In the remote invocation the client sends remotely a request to a provider asking the execution of a service. The execution takes place in the provider platform. In local invocation the client asks remotely to transfer a copy of the service to its device. After being transferred, the execution takes place in the client site.

5.5 The Cache Engine

The principle of the Cache Engine is to store expensive (time-consuming) information to create, so it can be automatically reused. For example if a composite service requires complex calculations, caching may save processing time, speed up the response rate for the client and lessening the burden on the service servers CPU. In other words the Cache Engine stores the resulting process model of a query as well as its context, and reuses it for other clients that supply the same query with the same parameters. Caching is also useful when the same client supplies the same query in other contexts; first the cache is checked if the same composite service would satisfy the new query needs (according to the new context), if not the composite service is decomposed, services that do not meet the new requirements are composed-out, and a new discovery/composition process is triggered for the composed-out services. The new discovered services are composed-in with the cached ones. Finally the resulting composite-service is recommended to the user, and a copy of the process model is kept in cache as well as its context. Another advantage of the Cache Engine

---

2 In our framework the strong definition of M-services is used.
Engine is to provide the system with a fault tolerance mechanism. We are aware that maintaining stable communication channels during the whole discovery/composition and execution process may not be possible in a highly dynamic and pervasive environment. For this means breakpoints are introduced in the execution process, and the broker for a particular request sends back the results after a sub-task is completed, these results are cached. When a failure occurs instead of making the BA initiating a new discovery/composition request, it reconstructs only the query that is still unsolved.

6 The SmartCadie Project

As a first example we mention the SmartCadie\(^3\) project. SmartCadie stands from SMart ARTicles in a Context-Aware Dynamic Identification Environment, (Fig. 4). The aim of this project is to assist handicapped, and blind persons with a portable interface that provide them with information of interest in a supermarket, example of such information can be: prices, directions, services availability, promotional messages and suggestions upon their context (i.e. location, profile, preferences...etc). The system is capable to detect users location and communicate through a wireless infrastructure with services providers around the area so as to gather data and information requested by the user. For this purpose ordinary market carts where equipped with embedded PCs that are linked to the Internet over an IEEE802.11b WLAN connection. Each cart is running a Lightweight-UDDI (LW-UDDI) to offer its services. Users may request services from their local LW-UDDI or trigger a service discovery process from the nearby LW-UDDIs hosted in other carts. Users may also trigger a service composition process to compose services out of the registered ones. Various multimedia means are used to keep users as much comfortable as possible, for example Non-Visual Interfaces are offered to blind persons to facilitate the interaction with the system.

This is a small set of our framework nevertheless demonstrates its benefits with respect to the contextualization of services discovery and composition in an indoor environment.

7 Conclusion and Future Work

In this paper, we present our vision and initial thoughts on a new direction of research for bringing into the common fold Context-aware computing and Web services. Our long

\(^3\) Caddie is the French translation of the word market cart.
term research objective is to allow users to satisfy their needs regardless of their location and the resources that are considered in the performance of the services. Our contribution is the enhancement of the service discovery and composition by taking into account the available contextual information. Our ongoing and future work covers two research thrusts. The first one concerns the extension of the system with a formal definition of context for services. We also intend to exploit context-awareness in order to improve existing Web services standards, such as WSDL and UDDI. The second research thrust concerns security. Since the information stored in the user profile is held on behalf of the user, it is essential that she controls the authorization given to other applications and services. It is also very important to enable users to switch off access to their context information to everyone or to some services and applications. All these security aspects are being studied within the CB-DA project [26] (A Generic Framework for Context-Based Distributed Authorizations) at the Software Engineering Research Group at the University of Fribourg in Switzerland.

References


