

Chapter XVI

Service Composition

Approaches for Ubiquitous and Pervasive Computing: A Survey

M. Bakhouya

George Washington University, USA

J. Gaber

Universite de Technologie de Belfort-Montbeliard, France

ABSTRACT

This chapter describes and classifies service composition approaches according to ubiquitous and pervasive computing requirements. More precisely, because of the tremendous amount of research in this area, we present the state of the art in service composition and identify key issues related to the efficient implementation of service composition platforms in ubiquitous and pervasive computing environments.

INTRODUCTION

Ubiquitous and Pervasive Computing (UPC) are new paradigms with a goal to provide computing and communication services all the time and everywhere. In **Ubiquitous Computing (UC)**, the objective is to provide users the ability to access services and resources all the time irrespective of their location (Gaber, 2000; Weiser, 1996).

Pervasive Computing (PC), often considered the same as **ubiquitous computing** in the literature, is a related concept that can be distinguished from **ubiquitous computing** in terms of environment conditions (Gaber, 2006). We can consider that the aim in UC is to provide any mobile device an access to available services in an existing network all the time and everywhere while the main objective in PC is to provide spontaneous services

created on the fly by mobiles that interact by ad hoc connections (Gaber, 2000, 2006)

In **ubiquitous and pervasive computing**, **service composition** plays a fundamental role, and automation will be essential to improve speed and efficiency of users' responses and benefits. **Service composition** is the act of taking several component products or services, and handling them together to meet the needs of a given user (Chakraborty, 2001). For example, in an online business process of reservation of air tickets, a reservation service carries out three distinct services: ticket availability check, credit card check, and updating the required database to reserve a ticket for the user. Therefore, these three services must be integrated together to serve numerous user requests (Oprescu, 2004).

In UPC, automatic **service composition** requires dealing with four major research issues: service matching and selection, scalability, fault tolerance, and adaptiveness. The service matching and selection is the first step in creating any composite service and requires a **service discovery** system. The role of **service discovery** system is to locate the service components that provide the functionality to be placed in the new service. It allows the selection of services providing functionalities (i.e., capabilities) that match the requested functionalities. More precisely, **service discovery** systems should also be able to find out all services conforming to a particular functionality, irrespective of the way of invocation. To achieve this requirement, semantic level reasoning of describing the functionality of service is required. This is why the **Web service** community has developed a number of languages to formally describe services in order to facilitate their discovery.

Recently, **Web services** are becoming the most predominant paradigm for distributed computing and electronic business. In other words, the Web has become the platform through which many companies communicate with their partners, interact with their back-end systems, and perform

electronic commerce transactions. Examples of **Web services** include bill payment, customized online newspapers, or stock trading services. As pointed out in Hu (2003), a **Web service** is a software system designed to support interoperable machine-to-machine interaction over a network. More precisely, **Web services** are self-contained, modular units of application logic that provide business functionality to other applications via an Internet connection (Bucchiarone & Gnesi, 2006). Several XML-based standards are proposed to formalize the specification of **Web services** to allow their discovery, composition, and execution (Baget, Canaud, Euzenat, & Saïd-Hacid, 2003; Zeng, Benatallah, Ngu, Dumas, Kalagnanam, & Chang, 2004). These standards are primarily syntactical; **Web service** interfaces are like remote procedure call and the interaction protocols are manually written. On the other side, the **Semantic Web** community focuses on reasoning about Web services by explicitly declaring their preconditions and effects with terms precisely defined in ontologies.

The **service discovery** system should also be scalable across large-scale networks and adaptable to dynamic changes especially when services dynamically join and leave the network. To implement this process, most **service discovery** systems like SLP (Guttman, 1999), Jini (Jini, 2000; Robert, 2000), and SSDS (Xu, Nahrstedt, & Wichadakul, 2001) require that service components must be stored in component directories that can be accessed at runtime. These centralized systems cannot meet the requirements of both scalability and adaptability simultaneously. Several decentralized systems are proposed to address these issues. A survey and classification of **service discovery** systems proposed in the literature are presented in Bakhouya and Gaber (2006b).

Service coordination and management is the second issue to be addressed in automatic **service composition**. More precisely, composition platforms must have one or some brokers that coordinate and manage the different services

involved in the composition. The problem of coordination and management becomes difficult when the entities are distributed across the network and poses a scalability problem, especially when numerous users are concurrently making composite service requests. For example, if elementary services are distributed across the network, the use of a centralized broker requires a lot of central processing unit (CPU) cycles to process users' composition requests.

Since a composite service is dependent on many distributed elementary services, fault tolerance is another important issue to be included in **service composition** platforms in order to ensure their proper functioning. More precisely, if an elementary service shuts down, the request processing is hindered (Chakraborty & Joshi, 2001). In this case, the platform should be able to detect and restore it. It should be noted also that in dynamic networks, where services are coming up and going down frequently, the **service composition** platform should be able to adapt its composition by taking maximum advantage of the currently available services. This increases the composite service availability in a dynamically changing environment.

These important issues can be classified into two major directions in dynamic **service composition** research. The first direction addresses languages to specify and describe services including complex planning mechanisms that utilize these descriptions to generate composite services. In other words, this research direction is trying to define languages to specify services, invocation mechanisms, and composite services. The second direction aims to develop architectures that enable scalability, fault tolerance, and adaptive **service composition**.

In this chapter, approaches proposed in the literature for **service composition** are presented. More precisely, the aim is to identify key issues related to the efficient implementation of a **service composition** platform for UPC. The first section will present the service-oriented architecture and

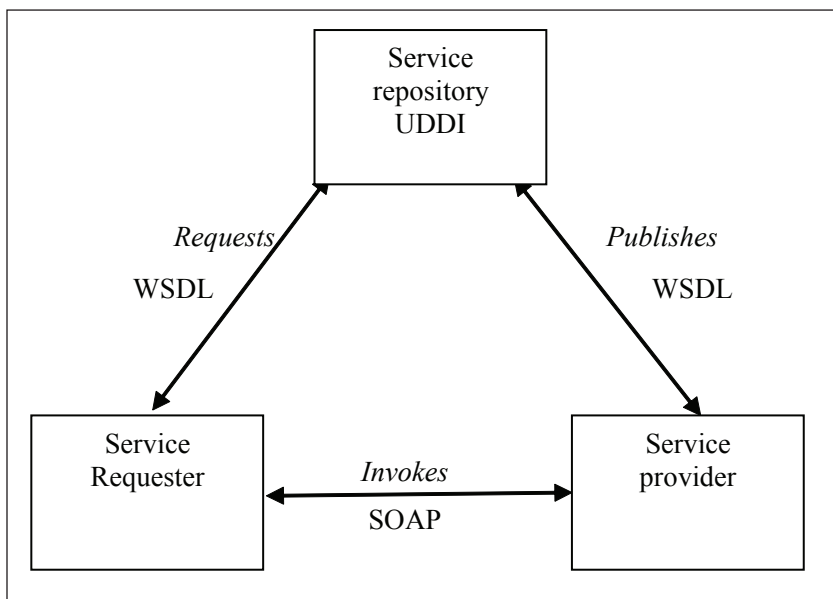
Web service generalities. In the second section, we will start from the current interest in **Web services** and compare the two major approaches: the **syntactical Web approach** and the **semantic Web approach**. In the third section, we will then explore the **Web service composition** methodologies and present the most **service composition** platforms known so far in UPC. It should be noted that automatic or dynamic **service composition** must involve the automatic selection, composition, and cooperation of appropriate services to perform and satisfy user task, given a high-level description of the task's objective. While in manual **service composition**, the user must select the required services, manually specify the composition, and ensure their interoperation in order to satisfy its request.

Service-Oriented Architecture (SOA) and Web Services

The current technological architecture for **Web services**, as depicted in Figure 1, is structured around three entities: the requesters, providers, and registries (Akram, Medjahed, & Bouguettaya, 2003; Dustdar & Schreiner, 2005; Wu & Chang, 2005). This architecture involves three major standards: Web Service Definition Language (WSDL), Universal Description, Discovery, and Integration (UDDI), and Simple Object Access Protocol (SOAP) (Maamar, Mostefaoui, & Yahyaoui, 2005). The aim is to define mechanisms to describe, advertise, bind, and trigger Web services. The service provider first publishes its WSDL services description in a UDDI registry. The service requester then searches in the UDDI for a **Web service** that matches the given criteria. If the required service is found, the requester invokes the service provider using SOAP messaging protocol.

UDDI is a repository, similar to a Jini repository (Robert, 2000), that provides mechanisms for service providers to publish their services and for clients to search required Web services. More

Figure 1. Web service entities with their roles and operations



precisely, using a UDDI interface, clients can dynamically look up as well as discover services provided by business partners. SOAP is a message layout specification that defines a uniform way of passing XML-encoded data. WSDL defines services as collections of network endpoints.

This **Web service** architecture is sufficient for some simple interaction needs; it is not sufficient for integration of business processes that involve multiple services (Padhye, 2004). More precisely, they do not deal with the dynamic composition of existing services (Serin, Hendler, & Parsia, 2003). For example, WSDL specifies only the syntax of messages that enter or leave a computer program (i.e., service), but not the order of messages (i.e., composition flow) that have been exchanged between services. A more challenging problem is to compose services dynamically in order to resolve unexpected user requests. This raises the need for **Web services composition** that provides the mechanism to fulfill the complexity of business processes execution. In other words, composition of **Web service** is needed to support

business-to-business or enterprise application integration (Srivastava & Koehler, 2003). This is why the business world has developed a number of **syntactical XML-based standards** to formalize the specification of composition and execution of composite services. However, these **Web service** flow specification languages, such as BPEL4WS (Curbera, Golland, Klein, Leymann, Roller, Thatte, & Weerawarana, 2002), are syntactical and do not deal with the semantic behavior of services. Today, Web services need to be described with additional semantic annotation to create a **Web semantic service-oriented architecture**, as depicted in Figure 2.

The **Web semantic** (Bucchiarone & Gnesi, 2006) is an extension of the current Web by marking up Web content, properties, and relations, in a reasonably expressive markup language with well-defined behaviors, for example, by declaring their preconditions and effects with terms defined in ontologies. In general, **ontology** is a shared conceptualization based on the semantic proximity of terms in a specific domain of inter-

est (Majithia, Walker, & Gray, 2004; Medjahed, 2004). In other words, the aim of ontologies is to extend the existing Web by adding a semantic level to its content. **Semantic ontology's languages** developed for this purpose are Darpa Agent Markup Language for Service description (DAML) and **Ontology** Web Language (OWL). Using these **semantic languages**, the information necessary to select, compose, and respond to services is encoded at the service Web site. The main objective is to develop **semantic languages** that will perform automatic composition by describing and exploiting services' capabilities and user constraints and preferences.

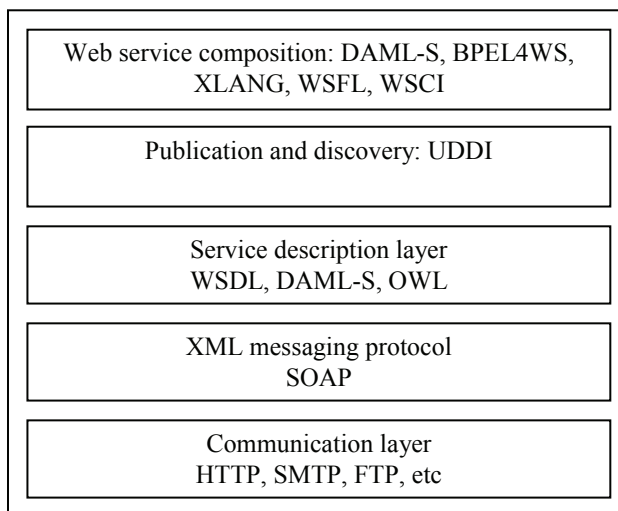
In the rest of this chapter, Web services description languages together with **service composition** models are presented. We start with the definitions of syntactic Web services and **semantic Web** services and then we present the **service composition** approaches.

SERVICES DESCRIPTION AND MODELING

Syntactic Description Languages

The **Web service** model evolves from three main XML-based technologies: UDDI, SOAP, and WSDL, as described above. WSDL is an **XML-based language** for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information (Milanovic & Malek, 2004). More precisely, it specifies the location of the service and communications that the service will perform. Four types of communication, called endpoints, are defined involving a service's operations (Bucchiarone & Gnesi, 2006): the endpoint receives a message (one-way) and sends a message (notification), the endpoint receives a message and sends a correlated message

Figure 2. Web service protocols and languages



(request-response), and it sends a message and receives a correlated message (solicit-response). Operations are grouped into port types, which describe abstract endpoints of a **Web service** such as a logical address under which an operation can be invoked (WSDL).

WSDL does not support semantic description of services like the definition of logical constraints between its input and output parameters. Generally, the major limitation of **XML-based languages**, such as WSDL, is their lack of explicit semantics that limit the capability of matching Web services. A semantic knowledge, using **ontology**, would help in the identification of the most suitable services for a particular request (Abela & Slanki, 2003). Ontologies play a key role in the semantic description extending syntactic description by providing precisely defined terms of services (Bucchiarone & Genesi, 2006). In other words, semantic description using ontologies will facilitate the automation of **Web service** tasks, including the automated **Web service discovery**, the execution, the composition, and the interoperation. The next section presents the most known semantic service description languages, DAML-S and OWL.

Semantic Description Languages

DAML-OIL, proposed by the Darpa Agent Markup Language (DAML) Committee, was one of the first languages designed for expressing ontologies. The service coalition of DAML has also developed a DAML-based **Web Service Ontology** to semantically describe **Web service**, namely DAML-S (Ankolekar et al., 2002). More precisely, DAML-S is a DAML-based **Web service ontology**, which supplies **Web service** providers with a core set of markup language constructs for describing the properties and capabilities of their services in unambiguous, computer-interpretable form. DAML-S is proposed primarily with the intention to provide the automatic **Web service discovery**, the automatic **Web service** invocation, the auto-

matic **Web service composition** and interoperation, and the automatic **Web service** execution monitoring. Automatic **Web service discovery** involves the automatic location of Web services that provide a particular service and that adhere to requested constraints. To realize this task, DAML-S provides declarative advertisements of service properties and capabilities that can be used for automatic **service discovery** (Narayanan & McIlraith, 2002). Automatic **Web service** invocation involves the automatic execution of an identified **Web service** by a computer program or a software agent. DAML-S provides declarative APIs for Web services that are necessary for automated **Web service** invocation. Automatic **Web service composition** and interoperation processes involve the automatic selection, composition, and interoperation of Web services to perform some tasks, given a high-level description of an expected objective. This task is realized by providing declarative specifications of the prerequisites and consequences of individual service use that are necessary for automatic **service composition** and interoperation. To realize the automatic **Web service** execution monitoring, DAML-S provides descriptors for the execution of services.

More precisely, to facilitate automatic **Web service discovery**, invocation, and composition execution monitoring tasks, DAML-S provides one possible representation through the class profile. This profile describes a service as a function of three basic types of information: the organization providing the service, the function that the service computes, and the service's characteristics. The provider information refers to the entity that provides the service (e.g., service name, contact information). The functional description of the service specifies the inputs (e.g., credit card number and expiration date) required by the service and the outputs (e.g., receipt) generated. It describes also the precondition required (e.g., valid credit card) by the service and the expected effects (e.g., card is charged) that result from the execution of the service. Finally, characteristics

of the service allow the description of the host properties that are used to describe features of the service (e.g., price).

In DAML-S, a service's process model, that describes the flow and data-flow involved in using a service, is also proposed. In this model, a service can be viewed as a process. A process has any number of inputs representing the information that is required for its execution. Under some condition, the process is executed to provide some number of outputs. In order for the process to be invoked, a number of preconditions must hold to generate a number of effects. In this way, a particular subclass of service model called process model is defined. Two leader components of a process model are also defined (DAML). The first, called process **ontology**, describes a service in terms of its inputs, outputs, preconditions, and effects. The second, called process control **ontology**, enables planning, composition, and service interpretation.

When defining processes using DAML-S, there are three tasks that need to be started for the process model to be successful. The first task relates process inputs to process' conditions, outputs or effects, including its precondition, the conditions governing conditional effects and conditional outputs, and the effects and outputs themselves. The second task relates inputs and outputs of a composite process to the inputs and outputs of its various subprocesses. The third task relates the inputs and the outputs of elements of a composite process definition to the parameters of other process components.

Ontology web language (OWL) has been developed by the W3C (World Wide Web Consortium). OWL has been inspired by DAML-OIL and provides three increasingly expressive sublanguages: OWL Lite, OWL DL (description logic), and OWL Full. OWL Full can be viewed as an extension of RDF (Resource Description Framework), while OWL Lite and OWL DL can be viewed as extensions of the restricted view of RDF. Recall that RDF is one of the first languages

for representing information about resources in the World Wide Web developed by W3C. The main difference between OWL Full and OWL DL lies in restrictions on the use of some of those features and on the use of RDF features. OWL Full allows free mixing of OWL with RDF Schema, like RDF Schema, and does not enforce a strict separation of classes, properties, individuals, and data values. OWL DL puts constraints on the mixing with RDF and requires classes, properties, individuals, and data values (OWL). To the contrary, OWL Lite is a minimal sublanguage of OWL DL, more practical, and easier to use for tools developers since it supports only a subset of the language constructs.

SERVICE COMPOSITION MODELING

As stated in the second section, the current **Web service** model, using WSDL, UDDI, and SOAP, enables the **service discovery**; however, it does not consider the automatic and dynamic integration and composition of services. More precisely, this model specifies only services and operations that perform, but not the order of, a flow specification of exchanged messages between services (Abela & Solanki, 2003). Several **Web service** flow specification languages like BPEL4WS (Business Processes Execution Languages for Web Services), XLANG, WSFL (Web Service Flow Language), and DAML-S have proposed to describe the order of messages to be exchanged between services (Akram et al., 2003; Curbera et al., 2002; Leymann, 2001; Van der Aalst, Dumas, & ter Hofstede, 2003).

In this section, **service composition** approaches are presented. As depicted in Figure 3, we can classify them into three categories: workflow-based approaches, **artificial intelligence (AI) planning-based approaches**, and **learning-based approaches**. More precisely, the purpose of this section is to provide a review of

these approaches, to help better understand how a Web services composition can be accomplished so far.

Workflow-Based Approaches

Recall that a **service composition** mechanism describes how different services can be composed into a coherent global service to satisfy the user request. As depicted in Figure 3, workflow-based approaches are classified into two categories: **orchestration methods** and **choreography methods** (Beek Bucchiarone, & Gnesi, 2006; Hu, 2003; Peltz, 2003a). **Orchestration methods** use a central coordinator (i.e., orchestrator) to combine and invoke Web services. More precisely, orchestration represents control from one party’s perspective and interactions occurring at the message level. The **choreography methods** do not assume the exploitation of a central coordinator. These methods define the conversation that should be undertaken by each participant. In other words, the composition is achieved via

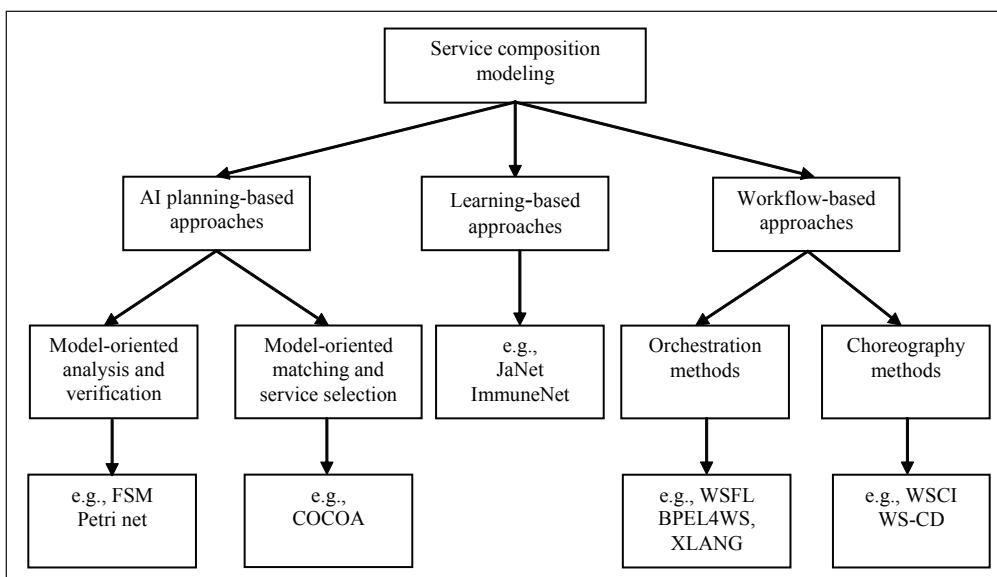
peer-to-peer interactions among the collaborative services. Hence, **choreography methods** are more collaborative than **orchestration methods** and allow each involved party to describe its part in the interaction (Hu, 2003).

Orchestration Methods

Several research projects regarding **service composition**, based on the orchestration principle, have been proposed. For example, the dynamic **service composition** called software hot-swapping (Menie & Pagurek, 2000) at Carleton University, eFlow (Casati, Ilnicki, Jin, Krishnamoorthy, & Shan, 2000) from HP Laboratories, and Ninja (Feng, 1999) **service composition** at the University of California Berkeley. These approaches typically address the fault-tolerance, scalability, and management of **service discovery** and composition requirements. They do not explicitly address languages issues to describe services.

EFlow, for example, is an e-commerce service platform that provides techniques for integrating

Figure 3. Classification of services composition approaches



various e-services to compose complex e-commerce transactions. The eFlow architecture consists of three entities (Chakraborty & Joshi, 2001): the eFlow composition, the **service discovery** system, and the elementary services. The role of eFlow engine is to schedule and maintain the state of every running composite service and coordinate the different events between e-services. The discovery system is responsible for finding required e-services. In eFlow, a composite service is modeled as a graph, which defines the execution order among the services. The graph is created manually and consists of service nodes that represent simple or composite services and event nodes or decision nodes that represent rules which control the execution flow between service nodes (Su & Rao, 2004). Each service is specified using XML language providing the some details of services such as URL to contact the service and input/output information.

More precisely, the eFlow is based a centralized broker which manages the **service composition** process, like path creation, **service discovery**, appropriate combination of different services, and management of the information flow between composed services. The drawback is that if a huge number of users attempts to access variety and increasing number of services distributed over the network, the broker quickly becomes a bottleneck.

Another interesting platform developed at Carleton University focuses on a dynamic **service composition** called hot-swapping (Feng, 1999; Mennie & Pagurek, 2000). Their work on **service composition** follows their research in the field of dynamic component up-gradation at runtime. More precisely, software hot-swapping is defined in Mennie and Pagurek (2000) as a process of upgrading software components at runtime in systems which cannot be brought down easily, cannot be switched off-line for long periods of time, or cannot wait for software to be recompiled once changes are made. The hot-swapping differs from other forms of software composition

since it deals exclusively with network services. Network services are individual software components, which can be distributed within a network environment, that provide a specific set of well-defined operations. This platform uses Jini (Jini, 2000) discovery architecture to locate services to be composed. **XML-based language** is used to describe capabilities, constraints, inputs, outputs, and dependencies of each service.

To carry out heterogeneous **service composition**, two techniques are proposed in Feng (1999): the interface fusion technique and the stand-alone technique. The first involves the formation of composite services that can be accessed by a common interface. The advantage of this technique is the speed at which a composite can be created and exported since it does not need to be constructed dynamically. With the second technique, a service is created by dynamically assembling the available services, for example, in the form of pipes and filters. In this configuration, the input to the composite service is sent to the first service component, which in turn sends its output to the input of the next service component and so on, along a chain. As stated by Feng (1999), the main advantage of this technique is that all service components do not need to share state information and they are not aware or dependent on other service components. However, the drawback of this technique is that the runtime composition could need a costly amount of time to construct and deliver a stand-alone composite service.

The Ninja platform has been developed to enable and to address dynamic **service composition** (Mao, Brewer, & Katz, 2001). It addresses the composition of arbitrarily complex services from simpler ones over a wide area network. The Ninja platform architecture is composed of the following five entities (Chakraborty & Joshi, 2001): Internet Service Provider (ISP), Network Service Providers (NSP), Automatic Path Creation (APC), Service Discovery Service Provider (SDSP), and End Clients (EC). ISPs are providers of resources that can be requested by clients. NSPs provide

connection to the various devices through varying types of networks. APC is the central entity that coordinates the **service composition** process. It enable service access, dynamic **service composition**, data flow optimization, and adaptability. SDSP is a discovery system similar to Jini or SLP directory systems responsible for locating required services available in the network using XML description language. ECs are machines, either fixed or mobiles, connected to the Internet. The **service composition** mechanism used in the Ninja platform is as follows. When an APC entity gets a request for a particular composite service, it figures the set of operators and connectors that help in transporting the data from one network operator to another. The APC creates a logical path by shortest path techniques (Mao et al., 2001). Then it creates, instantiates, and executes the corresponding physical path.

The Ronin Agent Framework (RAF) is a composition agent- and service-oriented architecture for deploying dynamic distributed systems (Chen, 1999, 2004). The main objective of this work regarding **service composition** is to enable mobile users to access all sorts of information from their mobile devices. The information is assumed to be available from static information providers residing in the fixed network infrastructure.

The main entity in the RAF is the notion of the Ronin agent and its corresponding deputy. The agent deputy acts as a front-end interface for the other agents in the system. To discover the different information about agents available in the system, an enhanced Jini Discovery framework, called Xreggie (Chakraborty, Perich, Avancha, & Joshi, 2001), is used. The Ronin agent framework includes an agent description facility based on the **syntactical language** XML to describe a service by defining its capability, and system feature on which it is running. Based on this description, the enhanced Jini lookup finds out a service that matches a request by comparing the XML attributes of each service and selecting one to the request.

Recently, the Xreggie system was enhanced using DAML language to describe the services and their capabilities. Consequently, this enhanced **service discovery** allows more flexibility in finding out different types of services and reasoning about their capabilities. Recall that DAML is a **semantic language** that is more powerfully expressive than XML and is seen as a very important language to enable the service selection and matching. More precisely, the main advantage is that it allows the discovery system to locate services by semantically reasoning about the request and service descriptions. However, the system suffers from the scalability imposed by the centralized Jini architecture. The **service composition** also depends on the centralized broker that represents the central point of failure. Also, the state information of each agent has to be maintained and the use of broadcast technique to disseminate the state information poses a scalability problem (Chakraborty, 2001).

Chackraborty and Joshi (2001) present a survey related to **service composition** platforms and evaluate criteria for judging protocols used to enable such composition. Their work claims that many of the current technologies still do not cover all these aspects in their implementation and deal with a centralized broker that manages the **service composition** process. The drawback is that if a huge number of users attempts to access a variety and increasing number of services distributed over the network, the broker quickly becomes a bottleneck, and in addition, it represents a central point of failure. Therefore, achieving coordination in collaborative applications that consist of composed Web services is difficult. It should be noted that the platforms presented above focus primarily on local collaborative applications without the emphasis on service description languages issues explicitly.

In parallel to these works which focus mainly on developing architectures that enable scalability, fault tolerance, and adaptive **service composition**, several service description languages that enable

developers to create, execute, and combine Web services into complex services using **Web service** model and language standards are proposed, for example, a Web Service Flow Language (WSFL) proposed by IBM to specify how to implement a business process model using the **Web service** architecture. In other words, WSFL is integrated with UDDI and WSDL for dynamic selection of Web services. Service providers must properly describe their services in order to be classified to handle a specific activity in the business process. More precisely, a business process is created using WSDL and is represented by a XML flow model. The flow model defines activities that are implemented in the form of Web services. It defines also the flow data sequences from an activity to another (Abela & Solanki, 2003).

XLANG is another flow composition language developed initially by Microsoft for the creation of business processes and the interaction between **Web service** providers (Abela, 2003; Satish, 2001). The specification provides support for sequential, parallel, and conditional process control flow. It also included a robust exception handling facility, with support for long-running transactions through compensation. XLANG uses WSDL as a means to describe the service interface of a process. More precisely, it extends WSDL language by adding some behavior capabilities. A behavior defines the list of actions that belong to the service and the order in which these actions must be performed. The execution order of XLANG actions is defined through control processes (e.g., sequence, while loop, etc.).

Recently, the Web services workflow specifications outlined by XLANG and WSFL have been superseded by a new specification from IBM, Microsoft, and BEA called BPEL4WS (Peltz, 2003a, b). BPEL4WS is a flow language that allows the specification of multiple Web services coordination. More precisely, BPEL4WS provides a notation for describing interactions of Web services as business process (Charif & Sabouret, 2005; Curbera et al., 2002). The definition of such busi-

ness protocols involves the precise specification of the mutually visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal implementation. In this way, the BPEL4WS process defines how multiple service interactions are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination.

BPEL4WS assumes that services are described using XML-based specifications like WSDL (Peltz, 2003a, b). The interaction between services is described in a business protocol specification language. For this purpose, BPEL4WS provides a programming language-like constructs (e.g., sequence, switch, while) as well as graph-based links that represent additional ordering constraints on the constructs (Srivastava & Koehler, 2003). It also provides a set of primitive activities like Invoke, Receive, and Wait. BPEL4WS are extended with two significant and complementary specifications: WS-Coordination and WS-Transaction for providing protocols that coordinate the actions of distributed applications (Abela & Solanki, 2003). More precisely, the WS-Transaction specification allows a composite service to monitor the success or failure of each individual, and the coordinated activity. The WS-Coordination defines a framework through which the composite service can work from a shared coordination context. BPEL4WS use of WSDL port information for service description is quite expressive with respect to process modeling constructs.

It is worth noting that these service flow languages focus on representing compositions where services, flow of the process, and the bindings between services are static and known *a priori* (Su et al., 2004; Sirin et al., 2003; Beek & Rao, 2006).

Choreography Methods

Choreography is defined in Kavantaz, Budett, Ritzinger, Fletcher, Lafon, and Bareto (2005) as a mechanism that allows constructing composi-

tions of **Web service** participants by explicitly asserting their common observable behaviors. More precisely, choreography defines the rules and interactions of collaboration between two or more services. Choreography is described from the perspectives of all parties and defines the complementary observable behavior between participants of collaboration.

WS-CDL (Kavantaz et al., 2005), for example, is a **choreography language** for specifying peer-to-peer protocols where each party is autonomous without hierarchy among them, and there is no centralization point like in **orchestration approaches**. More precisely, a WS-CDL choreography description is a collection of activities that may be performed by one or more participants. There are three types of activity in WS-CDL, namely control-flow activities (e.g., Sequence, Parallel, and Choice), work unit activities, and basic activities. Control-flow activities are similar to the basic control-flow constructs found in typical imperative programming languages. It can also be seen that these activities correspond to the Sequence, Flow, While, Switch, and Pick activities in BPEL4WS. A work unit describes the conditional and, possibly, repeated execution of an activity. The third type of WS-CDL activities, basic activities, describes points in a choreography where one role performs no action or performs an action behind the scenes that does not affect the rest of the choreography (Barros, Dumas, & Oaks, 2005).

To enable static validation and verification of choreographies to ensure that the runtime behavior of participants conforms to the choreography plan, WS-CDL must be based on a formal language that provides these validation capabilities (Barros et al., 2005). In addition, descriptions that abstractly specify behavior at a higher level, in terms of capability, would allow runtime selection of participants able to fulfill that capability, rather than restricting participation in the choreography to participants based on their

implementation of a specific WSDL interface or WSDL operations.

Web Service Choreography Interface (WSCI) is another XML-based description language proposed by BEA Systems, Intalio, SAP AG, and Sun Microsystems for choreographing message flow between Web services (Abela & Solanki, 2003; Arkin, Askary, Fordin, Jekeli, Kawaguchi, Orchard, et al., 2002). WSCI describes the flow of messages exchanged by a **Web service** in a particular process, and also describes the collective message exchange among interacting Web services, providing a global view of a complex process involving multiple Web services (Arkin et al., 2002). WSCI is a construct-based language that deals with the external observable rather than the internal definition of service behavior. This behavior is expressed in terms of temporal and logical dependencies among the exchanged messages, featuring sequencing rules, correlation, exception handling, and transactions. The advantage of this description is to enable developers, architects, and tools to describe and compose a global view of the dynamic of the message exchanged by understanding the interactions with Web services. However, it does not expose any form of semantics and therefore does not facilitate the process of automated composition.

Self-Serv framework (Sheng, Benatallah, Dumas, & Mak, 2002) can compose Web services and the resulting composite service can be executed in a decentralized dynamic environment (Benatallah, Dumas, Fauvet, & Paik, 2001). The execution of a composite service is not dependent on a central scheduler like Eflow (Abela, 2003), but rather on software components hosted by each of the providers participating in a **service composition**. More precisely, service providers participating in a composition process collaborate in a peer-to-peer manner to ensure that the schema and the control flow of the composition are respected.

Recall that to describe the control-flow of composite services, there are several existing pro-

cess-modeling languages as described above. In Self-Serv, a subset of statecharts has been adopted to express the control-flow of the composite service. States can be simple or composite. A simple state corresponds to the execution of a service, whether elementary or composite. Compound states on the other hand, contain one or several entire statecharts within them, thereby providing a decomposition mechanism. When a composite state is entered, its initial state(s) become(s) active. The execution of a composite state is considered complete when it reaches (all) its final state(s).

This approach clearly provides greater scalability and availability than a centralized one where the execution of a service depends on a central scheduler. Specifically, the responsibility of coordinating providers participating in a composite service execution is distributed across several lightweight software components hosted by the providers themselves.

SpiderNet is a platform that differs from the above works by providing fully decentralized efficient **service composition** solution (Xiaohui, Nahrstedt, & Yu, 2004). It focuses on addressing the challenge of scalable QoS and resource management issues, which is important for composing QoS sensitive distributed applications. More precisely, SpiderNet is a service oriented P2P system called P2P service overlay where peers can provide not only media files but also a number of application service components such as media transcoding and data filtering as well as application-level data routing.

In SpiderNet, new services can be flexibly composed from available service components based on the user's function and quality-of-service requirements. A service is defined as a self-contained application unit providing certain functionality. Service components are represented by a service graph, which collectively deliver advanced composite services to the end user. The link in the service graph is called service link that can be mapped on an overlay network path.

SpiderNet platform implements the decen-

tralized **service discovery** based on the Pastry distributed hash table (DHT) to efficiently locate services. The composition protocol is described as flows. Given the user's QoS/resource requirements, the source first generates a composition probing message, called probe. Peers process a probe independently and in parallel until the probe arrives at the destination in order to locate multiple candidate service graphs. When the destination collects the probes for a request, it then selects the best qualified service graph based on the resource and QoS and sends an acknowledge message along the reversed selected service graph to confirm resource allocations and initialize service components at each intermediate peer. Finally, the application sender starts to stream application data units along the selected service graph.

Another framework (Basu, Ke, & Little, 2002) addresses a distributed platform based on a hierarchical task-graph approach to enable **service composition** in mobile ad hoc networks. In this work, a composite service is represented as a task-graph with leaf nodes representing logical services and edges between nodes represent required data flows between corresponding services. A service is a functionality provided by a single device or by a federation of cooperating devices. The main purpose of this framework is to achieve the following goals. The first allows the construction of complex distributed services from simpler services, while the second provides a runtime discovery of devices and instances of services that are most suitable for executing the larger distributed application. The third goal provides the rapid adaptation to node and link failures due to the dynamic changes of the environment (e.g., the users' mobility).

For instantiating hierarchical task graphs and for handling disruptions in services due to mobility of devices, distributed algorithms have been proposed (Basu et al., 2002). **Service composition** is coordinated and managed by the source of the request. More precisely, this approach selects the source of the request as the composition manager

for itself (Chakraborty Perich, Joshi, Finin, & Yesha, 2002; Chakraborty, Yesha, Finin, & Joshi, 2005). After a service is composed on demand and used, its components retain their associations for a certain interval of time. If another user requests the service, he does not have to compose it on demand. However, this approach employs global broadcasting techniques that can increase load traffic in the network.

In Abela and Solanki (2003), authors have identified several requirements for Web **service composition** languages. The most important requires that a composition language has to be adequately expressive, and it has to have a well-defined and robust formal model in order to facilitate the automated composition of services. In other words, in order to dynamically compose Web services, languages modeling flow between services needs to have well-defined semantics. Flow composition languages like XLANG, WSFL, BPEL4WS, WS-CDL, and WSCI may be syntactically sound using WSDL (i.e., adds semantics on top of WSDL), however they lack semantics and expressiveness. More precisely, they do not expose any form of semantics and therefore do not facilitate the process of automated and dynamic composition, which is suitable for **ubiquitous and pervasive environments**.

A more challenging problem, when a functionality that cannot be realized by the existing services is required, the existing services can be combined together on demand to fulfill the request (Fujii & Suda, 2004; Maamar et al., 2005; Sirin et al., 2003). More precisely, problems related to **Web service** are how to specify them in an expressive manner, how to dynamically discover and compose them, and how to ensure their correctness (Beek et al., 2006).

Recently, artificial intelligence based approaches have been proposed in the literature. They take two major research directions to address these issues. The first concerns composite services behavior correctness. Indeed, interactions between service components come

up with several problems like messages that are never received and the behavior incompatibility of interacted services. Therefore, interoperation of the independent communicating component services should be granted; we will denote this issue in what follows by the first requirement, that is, services behavior correctness. The second research area addresses approaches that enable dynamic, runtime semantic **service discovery**, interaction and composition across the Web; we will denote this issue by the second requirement. These approaches, grouped into the AI planning approaches category depicted in Figure 3, are the subject of the next section.

AI Planning-Based Approaches

Recently, several techniques with unambiguous and formal semantics have been proposed in order to verify that a **service composition** process works properly. These techniques, known as **formal methods**, are generally used for the specification and the verification of complex systems. Among them, a variety of approaches based on state-action models (e.g., labeled transition system, timed automata, and Petri nets) and process models (e.g., p-calculus) are used to formally describe and reason about Web services conversation and composition (Beek et al., 2006). However, **formal methods** allow simulating and verifying the behavior of Web **service composition** at design time. Thus, the verification enables the detection and the correction of errors as early as possible, but they do not address the second requirement related to dynamic composition because they do not deal with the location of services or the recognition or the selection of those that match to the target service requested, thus creating the suitable composite service (Charif & Sabouret, 2005). Approaches using ontologies and augmented with formal techniques have been developed specifically to describe flow of **service composition**. These approaches allow the description of Web services' conversation, which

is an important requirement to achieve dynamic **service composition**.

Model-Oriented Analysis and Verification

There are several methods to translate **service composition** descriptions (e.g., OWL-S or DAML-S, BPEL4WS) to formal description using **formal methods** like Petri net and labeled state transition have been proposed. For example in Hamadi and Benatallah (2003) and Narayanan and McIlraith (2002), a Web **service composition** using Petri net algebra is proposed. In this work, a service behavior is considered as an ordered set of operations and has an associated Petri net that describes it. Operations are modeled by transitions, and the states of the service are modeled by places. The arrows between places and transitions are used to specify causal relations. After specifying composition with a Petri net, some algebraic properties, such as absence of deadlocks, could be proved.

Colored Petri Net (CP-net) is also used to tackle the reliability of **service composition** as proposed in Yang, Tan, and Xiao (2005). Generally, formal approaches aim to introduce much simpler descriptions and to model services in order to ensure verification of properties such as safety and liveness. In other words, describing services in such an abstract way lets us reason about the composition's correctness.

An approach proposed in Ankolekar et al. (2002) uses the situation calculus to model the composition process with their inputs, outputs, preconditions, and effects. Axioms in the situation calculus are mapped onto Petri net representation, which are then used to describe the semantics of the DAML-S control and constructs. In Foster, Uchitel, Magee, and Kramer (2003) modeling and verifying the composition of Web services workflows using the Finite State Processes (FSP) notation is proposed. To illustrate how these compositions are verified, workflow scenarios

described by BPEL4WS using message sequence charts, together with a model-checking tool to interactively verify the workflow properties are constructed. Recall that BPEL4WS is a language used to specify interactions between Web services.

An approach proposed in Koshikina (2003) describes the mapping rules for translating a BPEL4WS Process Model to a finite state automaton. The aim of this approach is to analyze composition processes in order to detect possible deadlocks. To achieve this, a process algebra called the BPE-calculus is introduced. It is a small language which captures all the BPEL features relevant to the analysis. This process algebra is modeled using a labeled transition system with a verification tool called the Concurrency Workbench. This tool allows us to verify many properties of BPE-calculus processes specified in a logic called μ -calculus (Koshikina, 2003).

Other approaches to verify business process are based on model checking techniques. For example, in Beek et al. (2005) authors describe how to use the SPIN model checker to verify **Web service orchestration** using Web Services Flow Language. In order to do the verification using SPIN, business processes are first translated into Promela, the specification language provided by SPIN.

It is worth noting that these approaches simulate and verify the behavior of Web **service composition** at design time. However, a few of these methods address automated composition where the end user or application developer specifies a goal and an "intelligent" composition engine selects adequate services and offers the composition transparently to the user (Mokhtar, Georgantas, & Issarny, 2005). More precisely, unlike the specification and the correctness verification of the composition process, the aim of these methods is how to identify candidate services, compose them, and verify how closely they match a request. In other words, the issue involved in the composition is how to select the most suitable services among

a lot of services in a distributed manner. Recently, an interesting formal based approach classified as model-based matching and service selection strategy is proposed in Mokhtar, Georgantas, and Issarny (2006). The following subsection focuses on the description of this strategy.

Model-Oriented Matching and Service Selection

Recall that dynamic composition allows the integration on the fly of a set of required services to perform a user request or a given target task. More precisely, the aim of dynamic composition is how to identify and select candidate services, compose them, and verify how closely they match a user request.

It should be noted that matching services is an important function of dynamic **service composition**. It allows the selection of services providing capabilities that are semantically equivalent to the capabilities requested by the target user task (Mokhtar et al., 2005). Several matching algorithms have been proposed in the literature (Ankolekar et al., 2002; Sycara, Paolucci, Ankolekar, & Srinivasan, 2003). These algorithms are classified in Mokhtar et al. (2005) into two categories: interface-level matching algorithms and process-level matching algorithms. In the first category, services and requests are described as a set of provided outputs and required inputs. Matching between a service and a request consists in matching all outputs and inputs of the request against all outputs and inputs of the service, respectively. This is the most used algorithm for matching **Semantic Web** services at the interface level in the literature (Kavantzias et al., 2005; Paolucci, Kawamura, Payne, & Sycara, 2002; Sycara et al., 2003). In contrary, the second category of matching algorithms, based on conversation description at the process level, provides a more precise matching, since the conversation description is richer than the interface description. More precisely, service

conversation description provides more information about the service's behavior.

Recently, a **CON**versation-based **service COM**position middleware (COCOA) using OWL/DAML-S language that supports dynamic **service composition** by using both interface-level matching (i.e., signatures matching) and process-level approaches (i.e., behaviors matching) to construct semantic composite services, is proposed in Mokhtar et al. (2006).

It should be noted, as described above, that DAML-S profile, process and grounding provide only the specification of **Semantic Web** services. More precisely, DAML-S, which defines a clear semantics for services description, is not provided with a tool or a means for composing dynamically the specific functionalities or actions desired. Indeed, the composite process description must be given *a priori* and cannot be built at runtime (Charif & Sabouret, 2005). In COCOA, this specification is complemented on one hand by an execution model that preserves the DAML-S semantics and, on the other hand, by an implemented computational architecture that enables and ensures dynamic, runtime semantic **service discovery**, interaction, interoperation, and composition across the Web.

COCOA is composed of two main algorithms, a **service discovery** algorithm (COCOA-SD) and a conversation integration algorithm (COCOA-CI). COCOA-SD allows the discovery and the selection of services as candidates to the composition. COCOA-CI performs the dynamic composition of the selected services. Using these algorithms, COCOA allows a user with a task description to execute it on the fly without any previous knowledge about the networked services (Mokhtar et al., 2006).

The first step in COCOA is the selection of the most suitable services to respond to the user requirements using COCOA-SD algorithm. More precisely, this algorithm that allows the construction of a task's behavior by using fragments of the service behaviors is performed in two phases

as follows. The first phase concerns the semantic operation matching that allows the selection of services that may be integrated together to compose the user task. To perform this phase, both the requested capabilities and those provided by services, are described using OWL-S as a set of IOPEs (Inputs, Outputs, Preconditions, Effects). More precisely, the requested and advertised capabilities are represented by a set of provided inputs and preconditions and a set of outputs and effects. COCOA-SD is used to match a requested service capability with a set of advertised ones. Two levels of matching have been defined, exact matching and weak matching, as proposed in Mokhtar et al. (2005, 2006). There is an exact or weak matching between a requested and a provided capability if all the inputs of the provided capability are matched against inputs of the requested one, and all the outputs of the requested capability are matched against outputs of the provided one. The difference between the two matching mechanisms is that in weak matching mechanism subsumption is supported between matched inputs and outputs. In addition, the weak matching is recognized between the two capabilities, if preconditions of the provided capability can be inferred from preconditions of the requested one, and effect of the requested capability can be inferred from effects of the provided one. However, in the exact matching mechanism, preconditions and effects of both capabilities should be equivalent.

The objective of the semantic matching phase is to compare semantically described operations involved in the task's conversation with those involved in the services' conversations. After matching the capabilities of the target task with those of the networked services, the **service discovery** algorithm selects those ones that will be useful for the composition.

The second phase concerns the conversation matching that filters and selects the most suitable services by comparing the structure of the task's conversation with those of selected services in the first phase. To do this, the mapping rules for

translating an OWL-S process model to a finite state machine are defined in this method and formalized as follows. An automaton is represented by the 5-tuple $\langle Q, S, \delta, S_0, F \rangle$, where Q is a finite set of states, S is a finite set of symbols that define the alphabet of the language the automaton accepts, δ is the transition function; that is $\delta: Q \times S \rightarrow Q$, S_0 is the start state, the state in which the automaton is when no input has been processed yet, and F a set of final states, a subset of Q (i.e., $F \subset Q$).

In COCOA, the symbols correspond to the atomic processes involved in the conversation. The initial state corresponds to the root composite process, and a transition between two states is performed when an atomic process is executed. Each process, either atomic or composite, that is involved in the OWL-S conversation, is mapped to an automaton and linked together with the other ones in order to build the conversation automaton. This is achieved following the OWL-S process description and mapping rules as described in Mokhtar et al. (2006).

COCOA-SD uses conversation descriptions to select service capabilities that are semantically equivalent to required capabilities of the user task. To perform this process, regular expressions that represent languages generated by the user task automaton and automata of selected services are used. More precisely, consider L the language generated by the extracted regular expression and by L_1, L_2, \dots, L_n the languages generated by the automata of the selected services S_1, S_2, \dots, S_n respectively. COCOA-SD selects all service S_i such that $L \cap L_i \neq \emptyset$.

Once the filtration phase is achieved by COCOA-SD, COCOA-CI integrates all the automata of selected services in one global automaton and selects one sub-automaton that correspond to task's conversation automata. More precisely, the COCOA-CI algorithm is involved into three steps. The first step is to connect the selected services' automata to form a global automaton by adding a new initial state. ϵ -transitions (ϵ is

the empty symbol) are also added in one hand to link this state to each of the initial states of the selected services and, on the other hand, to link each final state of the selected services with the new initial state.

The next step of the conversation matching phase is to parse each state of the task's automaton by starting with the initial state and by following the automaton transitions. The objective of this step is to find at each step of the parsing process an equivalent state to the current one in the task. More precisely, equivalence is detected between a task's automaton state and a global automaton state, when for each input symbol of the former, there is at least a semantically equivalent input symbol of the latter. The result of this step gives a list of sub-automata of the global automaton that behave like the task automaton. The third step consists of finding a sub-automaton, among this list, that behaves like the task's automaton. More precisely, this step verifies that for each transition set that corresponds to an atomic conversation, there is no ϵ -transition going to the initial state before this conversation is finished. ϵ -transitions that connect final states to the initial state of the global automaton mark the end of a service conversation. This step allows eliminating a list of sub-automata that do not verify the atomic conversation constraints and select arbitrarily one of those that behaves as the user task. An executable description of the user task that includes references to required services is generated and sent to an execution engine that executes this description by invoking the appropriate service operations.

This approach permits to the arbitrary selection of the resulting **service composition** as they all conform to the target user task (Mokhtar et al., 2005, 2006). However, it does not allow the selection of the most effective composition among the eligible ones. More precisely, it does not allow the selection of the optimal or the best one. In general, most research to date in **service composition** are based on a broker that chooses the composite service after calculating all the

candidates. The client/server strategy based on a broker is not scalable to large networks due to the large amount of available services. To set up **self-adaptive service composition systems**, **reinforcement learning** mechanism could be used (Gaber, 2000, 2006). The next section presents **reinforcement learning**-based approaches that permit the selection and the emergence of the most suitable composite service in a distributed manner without any central controller.

Learning-based Approaches for Service Composition and Emergence

In 2000, Gaber (2000) has pointed out that most research to date in **service discovery** and **composition** is based on the **traditional client-server paradigm** (CSP). He states that this paradigm is impractical in **ubiquitous and pervasive environments** and does not meet their related needs and requirements of adaptability to a dynamic environment, self-organization, and emergence. Therefore, Gaber (2000, 2006) has proposed two alternative paradigms to the traditional **client to server interaction paradigm** to design and implement **Ubiquitous and Pervasive applications**: the **adaptive services-to-client Paradigm** (SCP) and the **Spontaneous Service Emergence Paradigm** (SEP). The **adaptive Services/Client Paradigm** can be considered the opposite of CSP. Indeed, with the traditional CSP paradigm, the user initiates a request, should know *a priori* that the required service exists, and should be able to provide the location of a server holding that service. With the alternative SCP paradigm, it is the service that comes to the user. In other words, in this paradigm, a decentralized and **self-organizing middleware** should be able to provide services to users according to their availability and the network status. As pointed out in Gaber (2000), such a middleware can be inspired, for example, from a biological system like the natural immune system. More precisely,

unlike the classical client/server approach, each user request is considered as an attack launched against the global network. An **immune networking middleware** reacts like the natural immune system against pathogens that have entered the body. It detects the infection (i.e., user request) and delivers a response to eliminate it (i.e., satisfy the user request).

The second alternative SEP paradigm to the client/server one is more adequate for **pervasive applications**. It involves the concept of **spontaneous emergence service composition** that suits **pervasive environments**. More precisely, spontaneous services can be created on the fly and be provided by mobiles that interact by ad hoc connections (Gaber 2000, 2006). The **Spontaneous Service Emergence Paradigm** (SEP) could also be implemented by a natural system that involves self-organizing and emergence behaviors (Gaber, 2000, 2006). Natural and biological systems like the human immune system (Jerne, 1994; Watanabe, Ishiguro, & Uchkawa 1999) has a set of organizing principles such as scalability, adaptability, and availability that are useful for developing a networking model in a highly dynamic and instable setting (Gaber, 2000).

Recently, **agent-based approaches**, with self-adapting and self-organizing capabilities, have been proposed to implement SCP and SEP, respectively (Bakhouya, 2005; Bakhouya & Gaber, 2006a, b; Gaber, 2006). More precisely, these approaches, inspired by biological and natural systems, provide scalable and adaptive **service discovery** and **composition** systems for **ubiquitous and pervasive environments**. Approaches to implement SCP and SEP are presented in the next section.

JaNet

Itao, Nakamura, Matsuo, Suda, and Aoyama (2002a) have proposed a platform called Ja-Net for service **emergence** in large-scale networks. Ja-Net Architecture is motivated by the observa-

tion that the above desirable properties, such as scalability and adaptability, have already been realized in various large-scale biological systems like the bee colony. More precisely, Ja-Net defines a framework for developing large-scale, distributed, heterogeneous, and dynamic network applications. In this framework, a service is implemented by a collection of **distributed agents**, called cyber-entities. Ja-Net achieves built-in capabilities to create/emerge services adaptively according to user preferences (Itao, Nakamura, Matsuo, Suda, & Aoyama, 2002b). This is analogous to a bee colony (a network application) consisting of multiple bees (cyber-entities). Each cyber-entity implements a functional component related to its service or application. In addition, cyber-entity has simple behaviors such as migration, replication, reproduction, relationship establishment and death, and implements a set of actions related to a service that the cyber-entity provides.

Applications are provided through interactions of its cyber-entities. To provide an application, cyber-entities first establish relationships with each other and then choose cyber-entities to interact with based on their relationships. Strength of a relationship indicates the usefulness of the partner and dynamically adjusted based on the level of satisfaction indicated by a user who received the application.

In Ja-Net, a cyber-entity consists of three main parts: attributes, body, and behaviors. Attributes carry information regarding the cyber-entity (e.g., cyber-entity ID, service type, keywords, age, etc.). The cyber-entity body implements the service provided. Cyber-entity behaviors implement nonservice related actions such as migration, replication, relationship establishment, and death. Attributes, body, and behaviors of cyber-entities are described by XML and communicate using Speech Act-based Ja-Net ACL (Agent Communication Language). However, Ja-Net is not fault-tolerant to node failures. This is because, if a node fails or leaves the network, cyber-entities residing in it are destroyed and their state informa-

tion is also removed. Consequently, cyber-entities could be misinformed about the presence of some services at particular nodes. Thus, since cyber-entities collaboration requires global information state maintenance, this platform does not address dynamic environments such as mobile ad hoc networks where nodes are mobiles and could be partially connected.

Recall that **Pervasive Computing**, often considered the same as **ubiquitous computing** in the literature, is a related concept that can be distinguished from **ubiquitous computing** in terms of environment conditions. We can consider that the aim in UC is to provide any mobile device access to available services in an existing network all the time and everywhere while the main objective in PC is to provide spontaneous services created on the fly by mobiles that interact by ad hoc connections (Bakhouya & Gaber, 2007; Gaber, 2000, 2006).

ImmuneNet

Recently, **agent-based approach**, with self-adapting and self-organizing capabilities, has been proposed in Bakhouya (2005); Bakhouya and Gaber (2006b); and Gaber (2006) to implement the SCP paradigm for UC applications. This approach, inspired by the human **immune system**, provides scalable and adaptive **service discovery** and **composition systems for ubiquitous environments**. In this approach, servers are organized into decentralized multi-agent communities. More precisely, a community represents a composite service. A component service is composed of a set of hardware or software resources that users need to discover and select. Semantic languages, such as DAML-S, that enable their dynamic discovery and their composition, could describe these services. For example, for a punctual need, a user who would like to open video files or create video CD might need the following services: a video player, format transcoding software, the MPEG-4 codec (for his wireless laptop), video effect or

edge detection algorithms, and so forth. Since he does not know from which hosts he should get these resources throughout his current session, the user submits his multimedia composed service as a request, eventually with QoS requirements, to the resource discovery system that will process it according to the network status at that instant.

In this approach, the communities' organization based on the federation of **autonomous agents** is required to be self-organizing with inherent support for scalability and adaptability to user requirements and network changes. To address scalability, service agents (Sagent) that represent the resources of their corresponding servers in the network are organized into communities. More precisely, service agents establish relationships between them based on their affinity. Affinity corresponds to the adequacy with which two service agents could bind to create a composed service or to point out a similar service. To address the adaptability issue, affinity relationships between service agents are dynamic; the affinity values can be adjusted at runtime to cope with changes in the network.

The resource discovery and composition system proceeds as follows. It consists of two processes: service dissemination process and user request resolution process. The resource dissemination is a mechanism for server agents to learn about the existence of each other in order to create affinity relationships. In this process, when a server joins the network, it creates a **mobile agent**, to publish its service. A **mobile agent** initiates a random walk and moves randomly in the network. **Agents** can create replications (i.e., clones) of themselves, and consequently, they will learn about the existence of the all other Sagents in the network. In particular, when two Sagents detect an interest in common, a relationship is established and the two Sagents will group themselves into the same community. More generally, two Sagents have interest in common if their respective resources allow the creation of a composed service or if the Sagent holds a similar resource

or service. It should be noted that, regarding the agent cloning operation, the distributed algorithm proposed in Bakhouya and Gaber (2006b) is used to regulate and control dynamically the number of clones spawned in the network.

The request resolution process is the process by which the user or an application will be provided by the required service information. In this approach, to locate a single service or a composed service (i.e., a community), the user creates a **mobile agent**, called request agent. This agent initiates a random walk in the network until it meets an appropriate community that can resolve the request (i.e., an initial entry point of a community). In this community, the request agent uses and **adjusts affinity relationships** in three stages: request forwarding stage, result backtracking stage, and service execution stage. During the forwarding stage, the request agent guided by the affinity relationships seeks server agents from inside the community that can provide the required resources. By the **affinity adjustments** during this stage, a selected path from within the community graph will emerge as a response to the request. When all the required resources are discovered, the request agent starts the backtracking phase. During this phase, the path computed between an endpoint in the community and the initial entry point will be reinforced globally by secondary **affinity adjustments**.

After the result backtracking phase, the requester node has all the discovered service paths and can choose one of them based on its affinity values and initiates the request agent that starts the service execution phase. In this phase, the request agent moves from the initial point of the service path, via the intermediate nodes, to the final point of the community and links elementary services together in order to form a composite service. During this phase, request agent triggers the service methods provided by each node in the service path and transports the result to other nodes until it meets the last node. If one elementary service fails during the service session, the

Sagent that represents it in the preceding node will detect it and find a substitute and recompose the service path. Since the Sagent has several affinity relationships to other Sagents that are similar or complementary, it selects a required one and sends a request agent to link it up.

It is worth noting also that this multi-organization based on dynamic affinities supported by relationships provides a highly decentralized system while remaining adaptive in a dynamic environment. More precisely, this decentralized architecture offers a high degree of resilience against servers leaving the network. For example, when a server leaves the network, all peer relationships with other servers are removed without additional overhead since it does not rely on any hard overlay control structure. Also, a leaving server or a communication failure does not have impact on the resolution of user requests.

The second alternative paradigm to the client/server one for **service composition** that suits **pervasive environments** involves the concept of spontaneous emergence. This paradigm can be carried out also by an inspired natural immune middleware that allows the emergence of ad hoc services on the fly according to dynamically changing context environments such as computing context and user context (Gaber, 2000). In this model, organizations or groups of autonomous agents represent ad hoc or composed services. More precisely, **agents** correspond to the **immune system** B-cells. **Agents** establish relationships based on affinities to form groups or communities to provide composite services. A community of **agents** corresponds to the idiotypic network in the human immune system (Gaber, 2006).

The service emergence model is the following (Gaber, 2000). A service is represented by an agent. The service is characterized by the set of functionality that is provided, while the agent represents the coherent behavior and activities of the service and its capability of performing tasks as a component of an ad hoc network (i.e., an ad hoc coalition). More precisely, a service agent is described by a

set of states such as C = Initialization or context updating, M = Mature composition, I = Immature composition, A = Annihilated composition, E = Execution, T = Termination. The emergence and the behavior evolution of composed services depend on the agents' internal states, the agent-to-agent interactions, and the context sensed by the agents during their executions (Gaber, 2000). More precisely, agents interactions, or B-cells, act like the idiotypic network of the immune system by stimulation/suppression chain to provoke the emergence of an immune response against antigens, that is, the ad hoc context (e.g., Bakhouya & Gaber, 2006c).

More generally, **agents** together with their affinity relationships as a whole form a **propitient multi-agent system** (PMAS) (Bakhouya & Gaber, 2006a). A **propitient system** is a system with the ability to self-organize in order to adapt towards the most appropriate agent organization structures according to unpredictable changes in the environment. This emergent behavior is delivered as a result of agents-to-agents and agent-to-environment interactions that adapt until the system hits a most suitable affinity network (Gaber & Bakhouya, 2006). In other words, a **propitient multi-agent system** implements the **Service Emergence Paradigm** (SEP) for **pervasive applications**.

CONCLUSION

The design and development of **ubiquitous and pervasive applications** require alternative operational models to the traditional **client-to-server interaction paradigm**. **Adaptive services-to-client paradigm** and **spontaneous service emergence paradigm** are more adequate to **ubiquitous and pervasive computing environments**. **Service composition** systems based on these three paradigms and proposed in the literature are presented with emphasis on automatic and dynamic **service composition** approaches and on

self-organization and self-adaptation approaches to implement SCP and SEP. Self-adaptation, self-organization, and emergence are crucial issues in systems that operate in an open and dynamic environment.

FUTURE WORK

Service composition is an important and active area of research and has been addressed widely in the fields of **Web service** and **service-oriented architecture**. As stated above, research in **service composition** can be divided into two directions: the service model description languages and the **service discovery** and **composition platforms**.

Several languages such DAML-S have been developed to address the first requirement. Indeed, the **Web service** community concentrates its efforts mainly on developing Web services with rich semantic annotation to create a **Web semantic service-oriented architecture**. Hence, several service description languages that enable developers to create, execute, and combine Web services into complex ones using the standard **Web service** model have been proposed. It should be noted that the standard Web-based services model has a centralized architecture based on the traditional client-server paradigm (CSP). However, to meet UPC requirements, new design approaches for **Web-service discovery** and composition are required and should be scalable and adaptable to continuously changing UPC environments. Therefore, languages with a higher semantic level based on ontologies to facilitate services discovery and composition should be addressed.

In parallel to this issue, the service-oriented architecture community has focused mainly on developing architectures that enable scalable, fault tolerance, and adaptive **service discovery** and composition to fulfill the second requirement, but the development of semantic languages is not addressed. Moreover, most of the proposed systems are based on the client to server paradigm (CSP)

based on the systematic usage of repositories and registries. However, the ability to maintain, allocate, and access a variety of continuously increasing heterogeneous resources and services distributed over a network is difficult to achieve with the traditional client-server approaches (Gaber, 2000, 2006). More precisely, these architectures cannot meet simultaneously the requirements of scalability and adaptability suitable for **ubiquitous and pervasive environments**.

As mentioned in this chapter, an appropriate model, the **Adaptive Services-To-Client Paradigm** for **service discovery** and composition in UC has been proposed. In this paradigm, decentralized and **self-organizing middlewares** should be developed with the ability to provide services to users according to their availability and according to the network status at the user request moment. This kind of middleware should be intelligent, scalable, adaptable, and available in highly dynamic and instable environments.

The **Spontaneous service Emergence Paradigm** is the alternative paradigm to the traditional client-server one that suits **pervasive environments**. This paradigm involves the concept of **spontaneous emergence of services** without *a priori* planning. This paradigm can be carried out by middleware that allows the emergence of ad hoc services on the fly according to dynamically changing context environments such as computing context and users' context.

In UPC environments, approaches dedicated to identify and select candidate services, to compose them, and verify how closely they match a user request, should emphasize self-organizing and self-adapting principles that are crucial issues for systems that operate in an open and dynamic environment. To achieve these requirements, approaches and languages proposed by Web services and **service-oriented architecture** communities should be combined together with the main objective to develop highly flexible and scalable

approaches and platforms with self-organizing capabilities in order to cope with dynamically changing contexts and networks environments.

REFERENCES

- Abela, C. (2003). *Semantic Web Services Composition*. Retrieved August 23, 2007, from <http://www.citeseer.ist.psu.edu/735292.html>
- Abela, C., & Solanki, M. (2003). *A Landscape of Markup Languages for Web Services Composition NetObject*. Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/solanki03landscape.html>
- Akram, M. S., Medjahed, B., & Bouguettaya, A. (2003). Supporting Dynamic Changes in Web Service Environments. In *Proceedings of the International Conference on Service Oriented Computing* (LNCS 2910, pp. 319-334, ISBN: 978-3-540-20681-1).
- Ankolekar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D., McDermott, D., McIlraith, S. A., Narayanan, S., Paolucci, M., Pyne, T., & Sycara, K. (2002). DAML-S: Web Service Description for the Semantic Web. In *Proceedings of the First International Semantic Web Conference (ISWC)*. Retrieved August 23, 2007, from <http://www.cs.cmu.edu/~softagents/daml.html>
- Ankolekar, A., Huch, F., & Sycara, K. (2002). Concurrent Execution Semantics for DAML-S with Subtypes. In *Proceedings of the 1st International Semantic Web Conference on The Semantic Web* (LNCS 2342, pp. 318-332, ISBN: 3-540-43760-6).
- Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., & Zimek, S. (2002). *Web Service Choreography Interface (WSCl)*. Retrieved August 23, 2007, from <http://www.w3.org/TR/wsci/>

- Baget, J. F., Canaud, E., Euzenat, J., & Saïd-Hacid, M. (2003). Les Langages du Web Sémantique. *INRIA Rhône-Alpes*. Retrieved August 23, 2007, from <http://www.inrialpes.fr/exmo/cooperation/asws/ASWS-Langages.pdf>
- Bakhouya, M. (2005). *Self-adaptive Approach Based on Mobile Agent and Inspired by Human Immune System for Service Discovery in Large Scale Networks*. Unpublished doctoral thesis, Université de Technologies de Belfort-Montbéliard (UTBM).
- Bakhouya, M., & Gaber, J. (2006a). *Self-organizing Approach for Emergent Multi-agent Structures*. In *Proceedings of the Workshop on Complexity through Development and Self-Organizing Representations (CODESOAR'06) at GECCO'06*. Seattle: ACM Press.
- Bakhouya, M., & Gaber, J. (2006b). Adaptive Approaches for Ubiquitous Computing. In (Eds.), *Mobile Networks and Wireless Sensor Networks* (pp. 129-163, ISBN: 2-7462-1292-7). Hermes Science.
- Bakhouya, M., & Gaber, J. (2006c). Adaptive Approach for the Regulation of a Mobile Agent Population in a Distributed Network. In *Proceedings of the 5th International Symposium on Parallel and Distributed Computing (ISPDC'06)*, Timisoara, Romania. IEEE Press.
- Bakhouya, M., & Gaber, J. (2007). Ubiquitous and Pervasive Applications Design. In D. Taniar (Ed.), *Encyclopedia of Mobile Computing & Commerce*. Hershey, PA: Idea Group Publishing.
- Barros, A., Dumas, M., & Oaks, P. (2005). A Critical Overview of the Web Services Choreography Description Language (WS-CDL). *Business Process Trends*. Retrieved August 23, 2007, from <http://www.bptrends.com>
- Basu, P., Ke, W., & Little, T.D.C. (2002). Scalable Service Composition in Mobile Ad hoc Networks Using Hierarchical Task Graphs. In *Proceedings of the 1st Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2002)*, Sardegna, Italy. Retrieved August 23, 2007, from <http://hulk.bu.edu/pubs/publications.html>
- Beek, M., Bucchiarone, A., & Gnesi, S. (2006). *A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods* (Tech. Rep. Bib. Code 2006-TR-15). Retrieved August 23, 2007, from <http://dienst.isti.cnr.it/>
- Benattallah, B., Dumas, M., Fauvet, M. C., & Paik, H. Y. (2001). *Self-Coordinate, Self-traced Composite Services with Dynamic Provider Selection* (Tech. Rep. No. UNSW-CSE-TR-0108). The University of New South Wales Department of Computer Science and Engineering. Retrieved August 23, 2007, from <http://sky.fit.qut.edu.au/~dumas/>
- Bucchiarone, A., & Gnesi, S. (2006). *A Survey on Services Composition*. Retrieved August 23, 2007, from <http://www.selab.isti.cnr.it/ws-mate/program.html>
- Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., & Shan, M.-C. (2000). Adaptive and Dynamic Service Composition in eFlow. In *Proceedings of the Conference on Advanced Information Systems Engineering* (pp. 13-31). Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/casati-00adaptive.html>
- Chakraborty, D. (2001). *Service Composition in Ad-hoc Environments* (Tech. Rep. No. TR-CS-01-20). University of Maryland, Department of Computer Science and Electrical Engineering, Baltimore County, MD. Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/529431.html>
- Chakraborty, D., & Joshi, A. (2001). *Dynamic Service Composition: State-of-the-art and Research Directions* (Tech. Rep. No. TR-CS-01-19). University of Maryland, Department of Computer Science and Electrical Engineering, Baltimore

- County, MD. Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/chakraborty01dynamic.html>
- Chakraborty, D., Perich, F., Avancha, S., & Joshi, A. (2001). Dreggie: Semantic Service Discovery for M-commerce Applications. In *Proceedings of the Workshop on Reliable and Secure Applications in Mobile Environment, 20th Symposium on Reliable Distributed Systems* (pp. 28-31). Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/chakraborty01dreggie.html>
- Chakraborty, D., Perich, F., Joshi, A., Finin, T., & Yesha, Y. (2002). A Reactive Service Composition Architecture for Pervasive Computing. In *Proceedings of the 7th Personal Wireless Communications Conference (PWC 2002)* (pp. 53-62, ISBN: 1-4020-7250-3).
- Chakraborty, D., Yesha, Y., Finin, T., & Joshi, A. (2005). Service Composition for Mobile Environments [Special issue: Mobile Services]. *Journal on Mobile Networking and Applications*, 10(4), 435-451.
- Charif, Y., & Sabouret, N. (2005). An Overview of Semantic Web Services Composition Approaches. In *Proceedings of the International Workshop on Context for Web Services (CWS-05), ENTCS*, 146(1), 33-41.
- Chen, H. (1999). *Developing a Dynamic Distributed Intelligent Agent Framework Based on the Jini Architecture*. Unpublished master's thesis, University of Maryland, Baltimore County. Retrieved August 23, 2007, from <http://gentoo.cs.umbc.edu/ronin/doc/>
- Chen, H. (2004). *Ronin Agent Framework*. Retrieved August 23, 2007, from <http://gentoo.cs.umbc.edu/ronin/>
- Curbera, F., Golland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., & Weerawarana, S. (2002). *Business Process Execution Language for Web Services* (Version 1.0). Retrieved August 23, 2007, from <http://www-106.ibm.com/developerworks/library/ws-bpel/>
- DAML-S and Related Technologies. Retrieved August 23, 2007, from <http://www.daml.org/services/daml-s/0.9/survey.pdf>
- Dustdar, S., & Schreiner, W. (2005). A Survey on Web Services Composition. *International Journal on Web and Grid Services*, 1(1), 1-30. Inderscience Enterprises Ltd.
- Feng, N. (1999). *S-Module Design for Software Hot Swapping Technology*. Unpublished master's thesis, Carleton University, System and Computer Engineering Department. Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/feng99smodule.html>
- Foster, H., Uchitel, S., Magee, J., & Kramer, J. (2003). Model-based Verification of Web Service Compositions. In *Proceedings of the IEEE International Conference on Automated, Software Engineering* (ISBN: 0-7695-2035-9).
- Fujii, K., & Suda, T. (2004). Dynamic Service Composition Using Semantic Information. In *Proceedings of the 2nd ACM International Conference on Service Oriented Computing (ICSOC '04)* (pp. 39-48, ISBN: 1-58113-871-7).
- Gaber, J. (2000). *New Paradigms for Ubiquitous and Pervasive Computing* (Research Rep. No. RR-09-00). Université de Technologies de Belfort-Montbéliard (UTBM), France.
- Gaber, J. (2006). New Paradigms for Ubiquitous and Pervasive Applications. In *Proceedings of the 1st Workshop on Software Engineering Challenges for Ubiquitous Computing*, Lancaster, UK.
- Gaber, J., & Bakhouya, M. (2006). An Affinity-driven Clustering Approach for Service Discovery and Composition for Pervasive Computing. In *Proceedings of the ACS/IEEE International Conference on Pervasive Services ICPS'06* (pp. 277-280, ISBN: 1-4244-0237-9).

- Guttman, E. (1999). Service Location Protocol: Automatic Discovery of IP Network Services. *IEEE Internet Computing*, 3(4), 71-80. ISSN: 1089-7801.
- Hamadi, R., & Benatallah, B. (2003). A Petri-Net-Based Model for Web Service Composition. In *Proceedings of the 14th Australasian Database Conference on Database Technologies*, 17, 191-200. ISSN: 1445-1336.
- Hu, M. (2003). Web Services Composition, Partition, and Quality of Service in Distributed System Integration and Re-engineering. In *Proceedings of the XML Conference*, Philadelphia, PA. IDEAlliance. Retrieved August 23, 2007, from http://www.idealliance.org/papers/dx_xml03/papers/05-05-04/05-05-04.pdf
- Itao, T., Nakamura, T., Matsuo, M., Suda, T., & Aoyama, T. (2002a). Adaptive Creation of Network Applications in the Jack-in-the-Net Architecture. *IFIP Networking*, pp. 129-140.
- Itao, T., Nakamura, T., Matsuo, M., Suda, T., & Aoyama, T. (2002b). Service Emergence Based on Cooperative Interaction of Self-Organizing Entities. In *Proceedings of the IEEE Symposium on Applications and the Internet* (pp. 194-203), Nara, Japan.
- Jerne, N. (1974). Towards a Network Theory of the Immune System. *Ann. Immunol*, 125, 125-373.
- Jini. (2000). *Jini Network Technology: Specifications* (Version 1.1 Beta). Sun Microsystems Inc. Retrieved August 23, 2007, from <http://www.sun.com/software/jini/specs/>.
- Kavantzas, N., Budett, B., Ritzinger, G., Fletcher, T., Lafon, Y., & Bareto, C. (2005). Web Services Choreography Description Language Version 1.0. *World Wide Web Consortium*. Retrieved August 23, 2007, from <http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/>
- Koshkina, M. (2003). *Verification of Business Processes for Web Services*. Unpublished master's thesis, York University, Department of Computer Science, Toronto. Retrieved August 23, 2007, from <http://www.cse.yorku.ca/~franck/research/students/maria.pdf>
- Leymann, F. (2001, May). *Web Services Flow Language* (Rep. No. WSFL10). IBM Software Group.
- Maamar, Z., Mostefaoui, S. K., & Yahyaoui, H. (2005). Toward an Agent-Based and Context-Oriented Approach for Web Services Composition. *IEEE Transactions on Knowledge and Data Engineering*, 17(5), 686-697. ISSN: 1041-4347.
- Majithia, S., Walker, D. W., & Gray, W. A. (2004). A Framework for Automated Service Composition in Service-Oriented Architecture. *LNCS 3053*, 269-283. ISBN: 978-3-540-21999-6.
- Mao, M. Z., Brewer, E. A., & Katz, R. H. (2001). *Fault-tolerant, Scalable, Wide-area Internet Service Composition* (Tech. Rep. No. UCB//CSD01-1129). University of California-Berkeley. Retrieved August 23, 2007, from <http://www.cs.berkeley.edu/zmao/Papers/techreport.ps.gz>
- Medjahed, B. (2004). *Semantic Web Enabled Composition of Web Services*. Unpublished doctoral dissertation. Virginia Tech, Department of Computer Science. Retrieved August 23, 2007, from <http://www-personal.engin.umd.umich.edu/~brahim/>
- Mennie, D., & Pagurek, B. (2000). An Architecture to Support Dynamic Composition of Service Components, Systems and Computer Engineering, Carleton University. In *Proceedings of the 5th International Workshop on Component-Oriented Programming (WCOP 2000)*. Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/mennie00architecture.html>
- Milanovic, N., & Malek, M. (2004). Current Solutions for Web Service Composition. *IEEE Internet Computing*, 8(6), 51-59. ISSN: 1089-7801.

- Mokhtar, S. B., Georgantas, N., & Issarny, V. (2005). Ad Hoc Composition of User Tasks in Pervasive Computing Environments. *LNCS*, 3628, 31-46. ISBN: 978-3-540-28748-3.
- Mokhtar, S. B., Georgantas, N., & Issarny, V. (2006). COCOA: CONversation-based Service COmposition in PervAsive Computing Environments. In *Proceedings of the IEEE International Conference on Pervasive Services (ICPS'06)* (pp. 29-38, ISBN: 1-4244-0237-9).
- Narayanan, S., & McIlraith, S. A. (2002). Simulation, Verification and Automated Composition of Web Services. In *Proceedings of the 11th International World Wide Web Conference (WWW-11)*, Honolulu, HI (pp. 77-88, ISBN: 1-58113-449-5).
- Oprescu, J. (2004). *Découverte et Composition de Services dans des Réseaux Ambiants*. Unpublished doctoral thesis, Ecole Doctorale Mathématiques, Sciences et Technologie de l'Information, laboratoire LSR-IMAG. Retrieved August 23, 2007, from http://drakkar.imag.fr/article.php3?id_article=175
- OWL. Ontology Web Language. Retrieved August 23, 2007, from <http://www.w3.org/TR/owl-ref/>
- Padhye, M. (2004). *Coordinating Heterogeneous Web Services through Handhelds Using SyD's Wrapper Framework*. Unpublished doctoral thesis, Georgia State University, College of Arts and Sciences. Retrieved August 23, 2007, from <http://etd.gsu.edu/theses/>
- Paolucci, M., Kawamura, T., Payne, T., & Sycara, K. (2002). Semantic Matching of Web Services Capabilities. In *Proceedings of the 1st International Semantic Web Conference on The Semantic Web* (LNCS 2342, pp. 333-347, ISBN:3-540-43760-6).
- Peltz, C. (2003a). Web Services Orchestration and Choreography. *Computer*, 36(10), 46-52. ISSN: 0018-9162.
- Peltz, C. (2003b). Web Services Orchestration: a Review of Emerging Technologies, Tools, and Standards. *Hewlett Packard Co*. Retrieved August 23, 2007, from devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf
- Robert, E. M. (2000). *Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing* (Research Rep. No. UIUCDCS-R-2000-2154). Department of Computer Science, University of Illinois Urbana-Champaign, Urbana. Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/mcgrath00discovery.html>
- Satish, T. (2001). *XLANG Web Services for Business Process Design*. Retrieved August 23, 2007, from http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm
- Sheng, Q. Z., Benatallah, B., Dumas, M., & Mak, E. (2002). SELFSERV: 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. Retrieved August 23, 2007, from <http://sky.fit.qut.edu.au/~dumas/>
- Sirin, E., Hendler, J., & Parsia, B. (2003). Semi-automatic Composition of Web Services Using Semantic Descriptions. In *Proceedings of the Web Services, Modeling, Architecture and Infrastructure Workshop in conjunction with ICEIS 2003*. Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/sirin02semiautomatic.html>
- SOAP. Simple Object Access Protocol. Retrieved August 23, 2007, from <http://www.w3.org/TR/soap12-part0/>
- Srivastava, B., & Koehler, J. (2003). Web Service Composition: Current Solutions and Open Problems. In *Proceedings of the ICAPS 2003 Workshop on Planning for Web Services* (pp. 28-35). Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/srivastava03Web.html>

Su, X., & Rao, J. (2004). A Survey of Automated Web Service Composition Methods. In *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition*. Retrieved August 23, 2007, from <http://www.cs.cmu.edu/~jinghai/>

Sycara, K., Paolucci, M., Ankolekar, A., & Srinivasan, A. (2003). Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1(1), 27-46.

UDDI. Universal Discovery Description and Integration Protocol. Retrieved August 23, 2007, from <http://www.uddi.org/>

Van der Aalst, W. M. P., Dumas, M., & ter Hofstede, A. H. M. (2003). Web Service Composition Languages: Old Wine in New Bottles? In *Proceedings of the 29th EUROMICRO Conference* (pp. 298-305, ISBN: 0-7695-1996-2).

Watanabe, Y., Ishiguro, A., & Uchikawa, Y. (1999). Decentralized Behavior Arbitration Mechanism for Autonomous Mobile Robot Using Immune System. In *Artificial Immune Systems and Their Applications* (ISBN: 3-540-64390-7). Springer-Verlag.

Weiser, M. (1996). *Hot Topics: Ubiquitous Computing*. IEEE Computer.

WSDL. Web Services Description Language. Retrieved August 23, 2007, from <http://www.w3.org/TR/wsdl>

WSCl. *Web Service Choreography Interface 1.0 Specification*. Intalio, Sun Microsystems, BEA Systems, SAP.

Wu, C., & Chang, E. (2005). State-of-the-art Web Services Architectural Styles. In *Proceedings of the 3rd IEEE European Conference on Web Services (ECOWS)*, Vaxjo, Sweden. Retrieved August 23, 2007, from <http://wscc.info/p51561/files/paper54.pdf>

Xiaohui, G., Nahrstedt, K., & Yu, B. (2004). SpiderNet: An Integrated Peer-to-Peer Service Composition Framework. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC)* (pp. 110-119).

Xu, D., Nahrstedt, K., & Wichadakul, D. (2001). QoS-aware Discovery of Wide-area Distributed Services. In *Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, Brisbane, Australia (pp. 92-99).

Yang, Y., Tan, Q., & Xiao, Y. (2005). Verifying Web Services Composition Based on Hierarchical Colored Petri Nets. In *Proceedings of the 1st International Workshop on Interoperability of Heterogeneous Information Systems* (pp. 47-54, ISBN: 1-59593-184-5).

Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., & Chang, H. (2004). QoS-aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5), 311-327.

ADDITIONAL READING

Weiser, M. (1996). *Hot Topics: Ubiquitous Computing*. IEEE Computer.

Abela, C., & Solanki, M. (2003). A Landscape of Markup Languages for Web Services Composition. *NetObject*. Retrieved August 23, 2007, from <http://citeseer.ist.psu.edu/solanki03landscape.html>

Bakhouya, M., & Gaber, J. (2006). Self-organizing Approach for Emergent Multi-agent Structures. In *Workshop on Complexity through Development and Self-Organizing Representations (CODE-SOAR'06) at GECCO'06*. Seattle: ACM Press.

Service Composition Approaches for Ubiquitous and Pervasive Computing: A Survey

Bakhouya, M., & Gaber, J. (2007, February). Ubiquitous and Pervasive Applications Design. In D. Taniar (Eds.), *Encyclopedia of Mobile Computing & Commerce*. Idea Group Publishing.

Peltz, C. (2003). Web Services Orchestration and Choreography. *Computer*, 36(10), 46-52.