Washington University in St. Louis

Washington University Open Scholarship

All Computer Science and Engineering Research

Computer Science and Engineering

Report Number: WUCSE-2004-5

2004-03-10

Service Oriented Computing Imperatives in Ad Hoc Wireless Settings

Rohan Sen, Radu Handorean, Gruia-Catalin Roman, and Christopher Gill

Service oriented computing is a new paradigm that is gaining popularity in dis-tributed computing environments due to its emphasis on highly specialized, modular and platform-agnostic code facilitating interoperability of systems. It borrows concepts from more mature paradigms such as object-oriented and component computing. This results in a progression from object-oriented computing to component computing and finally to service oriented computing, a new paradigm for designing and delivering software. Just as an object encapsulates state and behavior at a fine level of granularity, a service offers similar encapsulation at a larger scale. This evolution raises the level of abstraction at... **Read complete abstract on page 2.**

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

Service Oriented Computing Imperatives in Ad Hoc Wireless Settings

Rohan Sen, Radu Handorean, Gruia-Catalin Roman, and Christopher Gill

Complete Abstract:

Service oriented computing is a new paradigm that is gaining popularity in dis-tributed computing environments due to its emphasis on highly specialized, modular and platform-agnostic code facilitating interoperability of systems. It borrows concepts from more mature paradigms such as object-oriented and component computing. This results in a progression from object-oriented computing to component computing and finally to service oriented computing, a new paradigm for designing and delivering software. Just as an object encapsulates state and behavior at a fine level of granularity, a service offers similar encapsulation at a larger scale. This evolution raises the level of abstraction at which systems are engineered, while preserving beneficial properties such as modularity, substitution and encapsulation. Every participant in a service oriented computing system is a provider or user of a service, or both. The service oriented computing paradigm is characterized by a minimalist philosophy, in that a user needs to carry only a small amount of code in its local storage, and exploits other services by discovering and using their capabilities to complete its assigned task. This chapter is the result of our experiences with designing and building service oriented computing frameworks for ad hoc wireless networks (Handorean & Roman, 2002). It examines the salient imperatives required to deliver a service oriented computing frame-work for ad hoc wireless networks. Ad hoc wireless networks are collections of hosts capable of wireless communication. Hosts within proximity of each other opportunistically form a network which changes due to host mobility. An ad hoc wireless network is a dynamic environment by necessity, which exhibits transient interactions, decoupled computing, physical mobility of hosts, and logical mobility of code. The network infrastructure is supported by the participating hosts themselves and there is no dependence on external, fixed resources. Ad hoc wireless environments are especially challenging to program when compared against other classes of fixed wireless environments because of the implications of mobility, i.e., frequent disconnections and inherent dynamism of the network on program execution. An important class of ad hoc mobile systems is based on small, portable devices, and this class of systems is the focus of this chapter. Such devices have limited storage capacity and battery power, which restricts the number of programs they can store and run locally. Service oriented computing offers a solution to this problem. By its very nature, service oriented computing is designed to facilitate sharing of capabilities while minimizing the amount of functionality a single host needs to maintain. Such a design is especially effective in ad hoc networks where storage space on individual hosts is at a premium, yet where the open environment allows a large number of hosts to contribute small functions resulting in a rich set of capabilities being available in the network as a whole. Service oriented computing has received much attention from researchers worldwide. However, most of this work has been focused on architectures and implementations for wired networks. Migrating service oriented computing to ad hoc networks is non-trivial and requires a systematic rethinking of core concepts. Many lessons have been learned from the work done in the wired setting, especially regarding description and matching of services. However, the more demanding environment of an ad hoc wireless network requires novel approaches to advertising, discovering and invoking services. We envision such ad hoc networks being used in a range of application domains, such as response coordination by firemen and police at disaster sites, or command and control of military units in a battlefield. Such scenarios demand reliability despite the dynamic nature of the underlying network. The motivation for this chapter is to understand the unique imperatives for a viable service oriented computing framework in ad hoc wireless settings, and to

illustrate selected solution strategies. We begin by examining current technologies, algorithms and capabilities that have been implemented for use in wired networks as a baseline. We then extend these concepts to cater to the special challenges of service oriented computing in ad hoc networks and direct the reader's attention to research issues in this area, presenting some of our own contributions in the process. The rest of the chapter is organized as follows. We describe existing service oriented computing architectures and the Semantic Web effort in the Background section. The section on Ad Hoc Wireless Network Perspective on Service Oriented Computing represents the main thrust of this chapter and discusses the elements of a service oriented computing framework, examining current technologies alongside our ideas on how these concepts may be applied to ad hoc networks. We cover potential areas of research in the Future Trends section. Finally, we summarize our findings in the Conclusion section.

Service Oriented Computing Imperatives
in Ad Hoc Wireless Settings

Rohan Sen, Radu Handorean, Gruia-Catalin Roman, and Christopher Gill

Department of Computer Science and Engineering
Washington University in St. Louis
Campus Box 1045, One Brookings Drive
St. Louis, MO 63130-4899, USA
{rohan.sen, radu.handorean, roman, cdgill}@wustl.edu

Introduction

Service oriented computing is a new paradigm that is gaining popularity in distributed computing environments due to its emphasis on highly specialized, modular and platform-agnostic code facilitating interoperability of systems. It borrows concepts from more mature paradigms such as object-oriented and component computing. This results in a progression from object-oriented computing to component computing and finally to service oriented computing, a new paradigm for designing and delivering software. Just as an object encapsulates state and behavior at a fine level of granularity, a service offers similar encapsulation at a larger scale. This evolution raises the level of abstraction at which systems are engineered, while preserving beneficial properties such as modularity, substitution and encapsulation. Every participant in a service oriented computing system is a provider or user of a service, or both. The service oriented computing paradigm is characterized by a minimalist philosophy, in that a user needs to carry only a small amount of code in its local storage, and exploits other services by discovering and using their capabilities to complete its assigned task.

This chapter is the result of our experiences with designing and building service oriented computing frameworks for ad hoc wireless networks (Handorean & Roman, 2002). It examines the salient imperatives required to deliver a service oriented computing framework for ad hoc wireless networks. Ad hoc wireless networks are collections of hosts capable of wireless communication. Hosts within proximity of each other opportunistically form a network which changes due to host mobility. An ad hoc wireless network is a dynamic environment by necessity, which exhibits transient interactions, decoupled computing, physical mobility of hosts, and logical mobility of code. The network infrastructure is supported by the participating hosts themselves and there is no dependence on external,

fixed resources. Ad hoc wireless environments are especially challenging to program when compared against other classes of fixed wireless environments because of the implications of mobility, i.e., frequent disconnections and inherent dynamism of the network on program execution.

An important class of ad hoc mobile systems is based on small, portable devices, and this class of systems is the focus of this chapter. Such devices have limited storage capacity and battery power, which restricts the number of programs they can store and run locally. Service oriented computing offers a solution to this problem. By its very nature, service oriented computing is designed to facilitate sharing of capabilities while minimizing the amount of functionality a single host needs to maintain. Such a design is especially effective in ad hoc networks where storage space on individual hosts is at a premium, yet where the open environment allows a large number of hosts to contribute small functions resulting in a rich set of capabilities being available in the network as a whole.

Service oriented computing has received much attention from researchers worldwide. However, most of this work has been focused on architectures and implementations for wired networks. Migrating service oriented computing to ad hoc networks is non-trivial and requires a systematic rethinking of core concepts.

Many lessons have been learned from the work done in the wired setting, especially regarding description and matching of services. However, the more demanding environment of an ad hoc wireless network requires novel approaches to advertising, discovering and invoking services. We envision such ad hoc networks being used in a range of application domains, such as response coordination by firemen and police at disaster sites, or command and control of military units in a battlefield. Such scenarios demand reliability despite the dynamic nature of the underlying network.

The motivation for this chapter is to understand the unique imperatives for a viable service oriented computing framework in ad hoc wireless settings, and to illustrate selected solution strategies. We begin by examining current technologies, algorithms and capabilities that have been implemented for use in wired networks as a baseline. We then extend these concepts to cater to the special challenges of service oriented computing in ad hoc networks and direct the reader's attention to research issues in this area, presenting some of our own contributions in the process. The rest of the chapter is organized as follows. We describe existing service oriented computing architectures and the *Semantic Web* effort in the Background section. The section on Ad Hoc Wireless Network Perspective on Service Oriented Computing represents the main thrust of this chapter and discusses the elements of a service oriented computing framework, examining current technologies alongside our ideas on how these concepts may be applied to ad hoc networks. We cover potential areas of research in the Future Trends section. Finally, we summarize our findings in the Conclusion section.

Background

In this section, we define the elements that make up a service oriented computing framework and then review some of the existing models that have been proposed using these elements. Finally, we highlight the *Semantic Web* as an example of an effort to establish and implement standards for service oriented computing in the wired setting.

Characterizing the Service Elements

A service oriented computing framework is a conglomerate of elements, each element fulfilling a very specific role in the overall framework. We list the salient elements required for a viable service oriented computing framework and the criteria used to judge their quality. We use this list as a basis for future discussions.

- The *service description* element is responsible for describing a service in a comprehensive, unambiguous manner that is machine interpretable to facilitate automation, and human-readable to facilitate rapid formulation by users. The aim is to specify the functions and capabilities of a service declaratively using a known syntax. A good service description mechanism must have a clear syntax and well defined semantics which facilitate matching of services at the semantic level.

- The *service advertisement* element is responsible for advertising a given service description on a directory service or directly to other hosts in the network. The effectiveness of an advertisement is measured as a combination of the extent of its outreach and the specificity of information it provides up front about a service, which can help determine whether a user would like to exploit that service.

- The *service discovery* element is the keystone of a service oriented computing framework and carries out three main functions. It *formulates* a request, which is a description of the needs of a user. This request is formatted in a similar manner to the service description. This element also provides a *matching function* that pairs requests with similar service descriptions. Finally, it provides a mechanism for the user to *communicate* with the service provider. A good discovery mechanism provides a flexible algorithm that matches advertisements and requests based on their semantics and maximizes the number of positive matches giving the user a more eclectic choice of services for his or her needs.

- The *service invocation* element is responsible for facilitating the use of a service. Its functions include transmitting commands from the user to the service provider and receiving results. It is also responsible for maintaining the connection between the user and the provider for the duration of their interaction. A good invocation mechanism abstracts communication details from the user and, in the case of network failure, redirects requests to another provider or gracefully terminates.

- The *service composition* element provides mechanisms to merge two or more services into a single composite service, which combines and enhances the functions of the services being composed. A good service composition mechanism leverages each of the elements listed above to provide automatic composition of services without human intervention.

Using Service Elements to Build Models

The elements described previously form the building blocks for most service oriented computing models. Various models entail the use of some or all of the elements described, depending on their complexity. We review some of the more popular models as background to our work and highlight features unique to each model. The *Service Location Protocol* (SLP) is designed for use on corporate LANs. *Universal Description, Discovery and Integration* (UDDI) and *Universal Plug and Play* (UPnP) are designed for use on the *Semantic Web*. *Salutation* is a general service model originally created with communication between appliances and other equipment in mind. *Jini* is a more general model that can be used on the web and with a wide range of applications.

The **Service Location Protocol** (SLP) (Kempf & Pierre, 1999) was developed by the SRVLOC working group of the Internet Engineering Task Force (IETF) with the idea of creating a service oriented computing standard that was platform independent for the Internet community. In SLP, every entity is represented as an agent. There are three kinds of agents - User Agents, which perform service discovery on behalf of clients, Service Agents, which advertise locations and capabilities on the behalf of services and register them with a central directory, and Directory Agents, which accumulate service information received from Service Agents in their repositories and handle service requests from User Agents. An interesting feature of SLP is that it can operate without the presence of a Directory Agent, i.e., it does not require a central service registry to function. User Agents search for a Directory Agent by default but if they do not receive any replies, they continue to interoperate directly with peer agents. If a Directory Agent starts operating at some point in the future, it advertises its presence and the User and Service Agents that receive the advertisement automatically start using the Directory Agent as a central service repository.

SLP registers services using templates which follow a specific pattern. Requests are made using a similar template though the parameters differ slightly. SLP's ability to operate in the absence of a Directory Agent makes it especially useful in ad hoc environments where it is not feasible to have a central service registry. Overall, SLP offers a clean model that is easy to conceptualize and, due to its design, it can be implemented readily in modern languages like C++ and Java.

Universal Description, Discovery and Integration (UDDI) (UDDI-Organization, 2000) was formulated jointly by IBM Corp., Ariba Inc., and Microsoft Corp. UDDI technology is aimed at promoting interoperability among web services. It specifies two frameworks, one to describe services and another to discover them. UDDI uses existing standards such as Extensible Markup Language (XML) (XML-Core-Working-Group, 2000), Hypertext Transfer Protocol (HTTP) (Fielding et al., 1999) and Simple Object Access Protocol (SOAP) (XML-Protocol-Working-Group, 2003).

The UDDI model envisions a central repository which is called the UDDI Business Registry (UBR). A simple XML document is used to specify the properties and capabilities

of a service. Registration with the UBR is done using SOAP. Information present in a service description can include things like the name of the business that provides the service, contact information for people in charge of administering the service, and identifiers for the service. The UBR acts as a mediator between businesses and services, and assigns a unique identifier to each business and each service. Marketplaces, search engines and enterprise level applications query the UBR for services, which they use to integrate their software better with other business entities like suppliers, dealers, etc.

The UDDI model is distinctive in that its approach looks at service oriented computing from a business process perspective. It envisions electronic interactions between businesses, resulting in trading of goods and services much like the business-to-business (B2B) model today but with enhanced capabilities and features. With the backing of industry giants like Microsoft and IBM, the future of this technology looks promising.

Universal Plug and Play (UPnP) (Microsoft-Corporation, 2000) was developed by Microsoft Corp. to facilitate seamless communication between networked devices in close proximity to each other. UPnP leverages TCP/IP and the Internet for its communication needs. It assumes a network that is dynamic, with devices frequently joining or leaving. A device, on joining a network, conveys its own capabilities upon request and learns about the capabilities of other devices. When it leaves the network, a device does so without leaving behind any explicit evidence that it was there. Entities in the network may be controllable devices or controllers. Controllers actively search for devices in proximity. The network is "unmanaged" in that there is no DNS or DHCP capability. Instead, a mechanism called AutoIP (Miller, 2003) is used to allocate IP addresses.

In UPnP a client, also called a *control point*, can undertake five kinds of actions. The discovery action is based on the Simple Service Discovery Protocol (SSDP) (Goland, Cai, Leach, & Gu, 1998). It exchanges information about the device type, an identifier, and a URL for more detailed information. During the description action, the control point uses the URL obtained during the discovery action to get a detailed XML description of the device. The control point then sends a control message encoded in XML to a Control URL using SOAP to discover the actions and parameters to which the discovered device responds. The event action consists of a series of events formatted in XML using the General Event Notification Architecture (GENA) (Cohen & Aggarwal, 1998) which reports changes in the state of the service. The presentation action consists of publishing a URL that can be retrieved by a control point, presumably in a format that allows an end user to control the discovered device. At this point UPnP assumes some external architecture and protocol exist to handle subsequent interactions with the device.

UPnP uses multicast for discovery and unicast for service utilization and event notification. Services and controllers are written using an asynchronous, multithreaded application model. UPnP requires a web server to transmit device and control descriptions, though this server is not required to be on the device itself. This requirement means that UPnP works best on the web and porting it to other environments, especially those with low-power hosts, is non-trivial.

Salutation (Salutation-Consortium, 2003) is an open standard service discovery and utilization model formulated by the Salutation Consortium. The vision for Salutation is not on the same scale as the world wide web. Instead, Salutation hopes to promote interoperability among heterogenous devices in settings such as corporate LANs where there is permanent connectivity from either wired networks or wireless gateways and disconnection is rarely an issue. In addition, Salutation strives to be platform, processor, and protocol agnostic.

Salutation has two major components (1) the Salutation Manager (SLM) which presents a transport independent API called the SLM-API and (2) The Transport Manager (TM) which is dependent on the network transport and presents an SLM-TI interface between the Salutation Manager and the Transport Manager. Services are registered via a local SLM or by a nearby SLM connected to a client. Service discovery occurs when SLMs find other SLMs and the services registered with them. Capability exchange is done by type and attribute matching. The SLM protocol uses Sun's ONC RPC (Srinivas, 1995). Periodic checks by the client ensure that it has the most current list of available services.

Salutation is interesting in that it solves a highly focused but widely relevant problem. An ideal environment for a Salutation deployment is a busy office space where there are a lot of computers, printers, fax machines and other electronic devices. Using Salutation to automatically discover and use devices within range eliminates the effort of individually configuring every single device. This makes Salutation a useful productivity enhancing tool.

Jini (Edwards, 1999) is a distributed service-oriented model developed by Sun Microsystems. Services in a Jini system can be hardware devices, software components, or a combination of the two. A Jini system has three types of entities; service providers, lookup services and users. A service provider multicasts a request for a lookup service or directly requests a remote lookup service known to it *a priori*. Once the handle to a lookup service has been obtained, the service provider registers a proxy object and its attributes with the lookup service. The proxy object serves as a client side handle to the actual service. The user makes a request for a service by specifying a Java type (or interface that the desired service must implement) and desired attributes. Matching is done based on type and values. Once a candidate service is found, the proxy object registered by the service provider is copied to the client using RMI (Sun-Microsystems, 2003b). Clients then use the proxy object to interact with the service.

The Jini model is designed for ubiquitous computing and is intrinsically scalable. It is language and protocol independent (though Java and TCP/IP seem to be natural choices for an implementation) and is resilient because multiple lookup services ensure that there is no single point of failure. Jini holds much promise for middleware developers that use the Java programming language, and has been formulated with Java-like languages in mind. Also, Jini is unique in that it introduces the idea of shipping a proxy object to the client where it is used as a handle to the actual service. The proxy approach allows complex services to reside on a server at a well known location. The service then simply

has to encode its well known location within its proxy, and can then be called from any client without that client having any knowledge of the actual location of the service. It also mitigates the issue of establishing the protocol between the service and the user since the proxy object hides all such communication from the end user.

These five models illustrate how various elements must come together and work as a cohesive whole to deliver the promise of service oriented computing. We now highlight the practical feasibility of such models by focusing on a model that promises to evolve the world wide web into a service oriented *Semantic Web*. While much work has already been done on this model, efforts continue to refine and augment its core.

Standardizing SoC for Wired Networks: The *Semantic Web*

The largest deployment of a service oriented computing model is the *Semantic Web* (Berners-Lee, Hendler, & Lassila, 2001). The Semantic Web effort aims to add logic to the current world wide web so that machines can infer the meaning and spirit of the content they handle. The vision is to evolve the web into a marketplace for services, where users and providers of services may conduct exchanges. The key technologies needed to deliver the *Semantic Web* concept are (1) innovative naming systems that provide simple constructs to describe complex objects, (2) relations that are machine interpretable, and (3) flexible matching algorithms that can match user requests with services based on the meanings of the request and service advertisement. Some issues of service oriented computing such as providing a central service registry and reliable links between user and provider are mitigated in this context, since one can leverage the existing infrastructure of the worldwide web. Hence the discovery and invocation elements are not especially significant in the web setting.

The *Semantic Web* uses a layered naming system. Lower level languages such as the Web Services Description Language (WSDL) (W3C-XML-Activity-On-XML-Protocols, 2003) describe *how* the data is sent across the wires. This layer handles all application layer protocol issues. Higher level languages describe *what* is being sent across and *why*. High level description languages are required to be clear and concise, and to support easy matching between two entities. Ontologies are high level languages that capture the semantics of an entity and its relation to other entities. An ontology is often itself structured into layers. DAML+OIL (Horrocks, 2002) is a combination of a markup language to construct ontologies (DAML) and an ontology inference layer (OIL) to interpret the semantics of the description. It is currently the language of choice for formulating ontologies for web services. The lowest layer of the system provides the syntax and is encoded in XML (XML-Core-Working-Group, 2000), since it is a W3C standard and hence has maximum acceptance in the market. Above the syntax layer a framework provides a basic set of constructs to describe entities and relations. In DAML+OIL, the Resource Description Framework (RDF) (W3C-Semantic-Web-Activity, 2003), also a W3C standard, fulfils this functionality. However, RDF is not powerful enough to describe entire ontologies. DAML+OIL fills this gap by extending the RDF concept to provide more constructs and relations. Finally, DAML-S (Ankolekar et al., 2002) is an ontology specific to web services that has been built using DAML+OIL.

DAML-S can be used to encode both service advertisements and service requests. The only remaining element is an algorithm that matches a service advertisement to a request. Conventional matching algorithms perform exact matches, which are considered too rigid for this purpose. Many inexact matching algorithms have been proposed to date, but due to constraints of space we discuss just one suggested by Paolucci et al. (Paolucci, Kawmura, Payne, & Sycara, 2002) for the *Semantic Web*. Their algorithm compares requests and advertisements and assigns a degree of similarity to each pair. Valid degrees of similarity are (from strongest to weakest): *exact, plugIn, subsumes* and *fail*. For every request, the algorithm returns the request-advertisement pair that has the strongest degree of matching. This allows inexact matches to be returned, which improves the flexibility of the system.

The *Semantic Web*, though still in its early stages, is a concerted effort towards introducing standards for service oriented computing in wired networks with the aim of unlocking its promise of unhindered interoperability between systems. In the next section, we explore the potential of service oriented computing in ad hoc wireless networks and discuss the imperatives, issues, technical challenges and benefits of enriching this new environment with flexible service oriented capabilities.

The Ad Hoc Wireless Network Perspective on Service Oriented Computing

In the previous section, we described the salient elements of a service oriented computing framework and reviewed some existing models. In this section, we revisit the same elements, but from an ad hoc wireless networking perspective. We discuss issues relevant to ad hoc networks and offer solutions in certain cases. From our discussion, we distill imperatives for service oriented computing in ad hoc wireless settings. To facilitate our discussion, we make use of an example and highlight how each element plays a unique and key role in the illustrated interaction.

Example: A Team of Surveyors

The Environmental Standards Compliance Group (ESCG) is responsible for ensuring that BigPower Inc., a power company with multiple power plants across the country does not violate any environmental standards. As part of their job function, ESCG is required to generate detailed maps showing the level of various pollutants in the soil surrounding every power plant. ESCG accomplishes this task by sending out teams of surveyors equipped with PDAs that collaboratively generate the required maps. Each PDA carries highly specialized software. For example, some PDA might carry software that interprets input from a GPS receiver while another PDA may carry a program that reads information from a soil probe.

Our example focuses on Robert, a member of the team of surveyors whose carries a PDA that has software that can map a set of given parameters. Robert's task today is to measure the level of pollutants in the soil. To achieve this, Robert pre-programs a query for a service that can provide soil analysis data and a service that can provide location data. When the team meets at a staging point, Robert's PDA executes the query and reports that David's PDA has a soil analysis data service. Robert keys in the names of the chemicals that he is interested in. Next, his PDA verifies that Sam's PDA can provide a GPS data service. However, Robert is only interested in the location and time stamp from the GPS data. He directs his PDA to filter out the extraneous data - the request is transmitted to the GPS data service on Sam's PDA which appropriately customizes the data sent to Robert. Data starts flowing in from both Sam's and David's PDA. By correlating the time stamp from the GPS data and the time stamp on the soil data, the software on Robert's PDA generates an accurate table of chemical levels and various locations. The remaining task is that of generating a graphic. Since Robert's PDA does not have a graphic tool, he enters a request for a map generating service. His PDA reports that Tom's PDA has a service that can take a data table and generate a graphic. Robert then directs his PDA to push his generated data to Tom's PDA which responds with a map in JPEG format which can be viewed by Robert on his PDA. Thus, using a complex set of services, Robert was effectively able to generate the required map with relative ease. In the following subsections, we show how each element fulfills a portion of the functionality required to deliver such services to users.

Description

The description element is responsible for dealing with issues related to creating a *profile*, a comprehensive and unambiguous description of a service. A comprehensive description of a service ensures the service can be used to the maximum extent of its capabilities. The unambiguous characteristic of the service description will ensure consistent interpretation of the data in the profile. The profile must be machine readable to facilitate the automation of the search process.

To make a profile machine readable, a well defined syntax must be imposed on the service description. Some service discovery infrastructures use simple data structures to describe their service profiles but most use markup languages like the Resource Description Framework (RDF) (W3C-Semantic-Web-Activity, 2003) and DAML (Horrocks, 2002). XML (XML-Core-Working-Group, 2000) is used for syntactic purposes. In some cases, the service profile can be replaced with an introspective analysis of the code. Using a technique called reflection, the code of a service can be inspected and its capabilities inferred from the list of methods it contains and their parameters. This technique is useful in languages that have reflection built in, as this mitigates the overhead of formulating a service profile.

Once a common syntax for the description has been established, the information in a service description can be broken into tokens and automatically processed by algorithms. However, the tokens have no value unless they have semantics associated with them. Se-

mantics can be defined by introducing a type system and establishing relationships between entities. RDF (W3C-Semantic-Web-Activity, 2003) describes relations between entities using directed graph-like data structures. Its Resource Description Framework Schemas (RDFS) (W3C-Metadata-Activity, 2000) extension provides additional basic notions such as classes. Details of the protocol to communicate with the service, if required, can be described using languages like WSDL (W3C-XML-Activity-On-XML-Protocols, 2003).

Syntax and semantics are combined in ontologies. Ontologies define specific terms for certain contexts and encode relationships between elements of a service description that help in matching the client's request to the right service profile. For example, the "Environmental Survey" ontology may have the `SoilData` class be a child of the `EnvironmentalData` class but the "GPS Location Provider" ontology may not even contain the `EnvironmentalData` class. The parties involved in the transaction need to make sure they understand the same semantics from the information represented in the service profile, i.e., they need to share a common ontology. There is an ongoing effort to develop ontologies for specific fields of activity building up a global scale standardization system. The reader is directed to http://www.daml.org/ontologies/ for samples of such ontologies.

The mechanics of describing a service essentially do not change when we move to ad hoc networks. Certainly the syntax can be reused and the semantics remain the same. However, it would be remiss to say that a service profile tailored for use in a wired network would work as effectively in an ad hoc environment. This is because the ad hoc network is a more demanding and dynamic environment. The service profile has to be enhanced to account for these differences. Examples of enhancements include adding location information, motion profiles (if the service provider is mobile), possible alternate providers in case a provider moves out of communication range, battery power remaining, trust certificates and other such details.

A example of an enhancement to a service profile for ad hoc networks is the addition of a motion profile. In the mobile ad hoc networking environment, motion information is an important component of a service profile since having the motion profile of a service provider can help schedule interactions, maintain or resume connectivity, schedule message delivery, etc. With this in mind, we developed a set of algorithms that compute windows of opportunity (Handorean & Gill, 2004) by comparing the motion profiles of the client and the service provider. Windows of opportunity represent a set of time intervals during which the client and service provider can interact (most likely because they are co-located or are within wireless communication range). During this time, they can schedule data or code exchanges. They can also use the information provided by the windows of opportunity algorithm to plan the nature of future interactions, which may include suspending computations while disconnected or choosing to delay an interaction to avoid unexpected termination.

Distilled Imperatives for Description in ad hoc settings
- The service profile must be comprehensive, unambiguous and machine interpretable.
- A common ontology is required to ensure that both user and provider infer the

same semantics for an entity.

    • The profile should be enhanced with details pertaining to the mobility and available resources of the service provider.

Advertisement

In our example, Robert's PDA automatically discovered that Sam's PDA provided a GPS location service and that David's PDA provided a soil analysis data service. This was a two-way process by which Robert's PDA requested such services and the services advertised themselves. In this subsection, we discuss issues pertaining to advertising a service. Simply put, advertisement is the process of putting the service profile and its ancillaries in a place where it can be read by anyone, so that users are made aware of the services' existence and can use them when they need to.

Advertisements come in many different flavors, ranging from simple text to complex objects. The simplest kind of advertisement is a plain text description of a service, e.g., encoded in XML (XML-Core-Working-Group, 2000). This kind of advertisement is used widely on the *Semantic Web*. Next in order of complexity is the two-step form of advertisement used by the UPnP (Microsoft-Corporation, 2000) system. The first step conveys information about the existence of the service and offers a Uniform Resource Identifier (URI) for a more comprehensive description. On obtaining this URI, the user connects to the specified location to get a complete description. The most advanced kind of advertisement is the proxy object approach used by the Jini (Edwards, 1999) system. The proxy object is advertised along with the service profile text. The proxy object behaves like a client side handle to the provider of the service and is used to control interactions between the user and provider.

Services may advertise themselves in two ways: on a central service registry or directly to users in a peer-to-peer fashion. The central registry is easier to maintain and configure. However, this design may introduce single points of failure. In such a setting, the complete failure of a lookup service could paralyze an entire community of hosts trying to interact. Another issue inherent in central directories is services that become suddenly unavailable (e.g., due to their host crashing or disconnecting) but which leave behind their advertisements in the registry. Lease agreements for services can be used to mitigate this issue but do not eliminate it. Finally, using a central registry means that the location of this registry may need to be disseminated using other channels. Using our example, Robert would have to know how to contact a local broker to obtain the address of the registry and would have to configure his PDA before he could discover the soil analysis service. While this is not unreasonable, it does add an extra step to the process of discovery.

The second approach to advertisement is the registry-less technique used by SSDP (Goland et al., 1998). Clients and servers advertise their needs and/or presence directly using unicast (when the location of the party is known), or multicast. The announcements happen when the process comes up, and possibly periodically thereafter. The advantage of this approach is that it eliminates the need for a third party mediator. Orphan

advertisements cannot exist due to the simple nature of the mechanism. The downside of the approach is the higher bandwidth usage and the limited scope of users an advertisement can reach. In the context of our example, the GPS location service would advertise itself individually to Robert's PDA and everyone else in proximity, whether they asked for such a service or not. Hybrid approaches exist that use both service directories and direct announcements. Users would then try to contact a service registry manager process and, if it were not available, they would start interacting in a peer-to-peer fashion.

An effective advertisement mechanism for the ad hoc setting needs to consider two main issues. The first is the size of the advertisement. Advertisements need to be as small as possible so that they do not monopolize bandwidth and can be transmitted in the short periods of time in which connectivity may exist. The second is the manner of advertising. Services should be advertised in a transiently shared space that adapts to host mobility. Neither of the approaches discussed previously, e.g., central registry or peer-to-peer are suitable. By nature of the network, a central well-known registry is not feasible and direct peer-to-peer advertisements use up a lot of bandwidth and battery power on resource-poor hosts.

A solution to the problem lies in the use of global virtual data structures. LIME (Murphy, Picco, & Roman, 2001) is a Java implementation of the Linda (Gerlenter, 1985) coordination model, adapted for ad hoc networking. The mechanism used for advertising services is based on transient sharing of a local service registry (Handorean & Roman, 2002) (Storey, Blair, & Friday, 2002). The content of the service registry is distributed across participating hosts and processes. Each process has its own local registry which it shares with local registries held by other processes, forming a federated service registry. The content of the federated registry is atomically updated as processes join or leave a defined community. When a process leaves the community, the advertisements in its local registry become inaccessible but the remaining processes are not affected. The approach guarantees the consistency of the service registry content at all times in that a service cannot be discovered if it is not available. Returning to our example, the GPS location service would put its service advertisement in its local registry. When Robert turned on his PDA, the local registry on Robert's PDA would merge with the GPS location registry and exchange data. Once the data transfer was completed, the two registries could separate.

DISTILLED IMPERATIVES FOR ADVERTISEMENT IN AD HOC SETTINGS
- The size in bytes of the advertisement should be minimized.
- Advertisements should be placed in a transiently shared space that is accessible by all interacting parties.
- The advertisements in the registry should be consistent and there should be no advertisements without there being an associated active service.

Discovery

The process by which a user finds a service provider is called service discovery. Service providers do the best they can to get their advertisements out to interested

users. However, as mentioned in the previous subsection, there is essentially a two-part process of (1) advertising by the provider and (2) requesting by the user. Without the user's request, there is no relation established between the user and provider. Requests are also referred to as *templates* that describe the required characteristics of a service.

Like advertisements, requests also have varying levels of complexity. The simplest kind of request contains the identifier of a service the client already knows. This scenario assumes that the client knows the protocol and the semantics of interacting with the provider and only needs to discover the provider's presence. When the client runs for the first time in a new environment or needs to search for a service it has never used before, the request becomes more detailed. Such a request is formulated as a template that describes the capabilities and performance parameters that the service must provide, in the form of attribute-value pairs. The formulation of a request follows the same guidelines that apply to the description of a service. A more advanced request breaks up the required list of attributes into "must have" and "can have" attributes. A candidate service is rejected if it does not have any of the "must have" attributes while the "can have" attributes behave like bonuses and are used as tie-breakers by the matching algorithm.

Like a service advertisement, a service request can be broadcast or multicast as repetitive or periodic messages. Indirect requests unicast their messages to a central service registry and obtain results from there. Models such as Salutation (Salutation-Consortium, 2003) support service brokers which are processes that manage service registries. The servers contact these brokers to advertise their services while the clients contact them to address their requests for services. If the model is based on service repositories and service brokers (lookup services), both clients and servers need to discover these special services first.

The most critical piece of the discovery mechanism is the algorithm that matches requests to advertisements. Matching algorithms themselves constitute a comprehensive research area: due to limitations of space, we mention them only briefly here. Matching can take two forms: *exact matching* and *inexact matching*. Exact matching is based on the logical equivalence operator. Exact matching can be a value match, where values of an object are compared, e.g., String greeting = "Hello" matches String salutation = "Hello" since their values are equivalent. Type matching is also a form of exact matching where the objects "greeting" and "salutation" are equivalent since they share the same type, i.e., "String." Exact matching is the kind of matching that is most commonly in use today. Inexact matching seeks to discover essential commonality of features of the two entities being matched. Even when two entities are not equivalent, a positive match is returned if the differences do not matter. This is a nontrivial process since teaching a machine to make correct judgements calls is a very complex task. The simplest kind of inexact matching is polymorphic matching. Polymorphic matching returns a positive result if one object is a subclass of the other. For example, a polymorphic comparison between a `LinkedList` object and a `DoubleLinkedList` object would return a positive match since `DoubleLinkedList` was a subclass of `LinkedList`. A more advanced kind of inexact matching called semantic matching is discussed later in this section.

An exact matching policy is in many circumstances considered too rigid for the purpose of matching services. Consider our example where Robert's PDA made a request for "Soil Analysis Data services" whereas the advertisement was for a "Constituents of Soil service." An exact matching algorithm would not return a match, but in fact the two are a good match for each other even though they do not use the same terms. A first step towards a more general matching policy is to allow for polymorphic matching, e.g., a client looking for a soil analysis service could be given a handle to a soil moisture content service if a soil moisture content service is in fact a subclass of a soil analysis service. Another type of inexact matching is inexact value matching or approximation, e.g., Robert may be looking for a soil moisture service but would accept a comprehensive soil data service if the soil moisture service were not available. The client can specify a range or set of values for a certain attribute rather than a single rigid value. It is also important for the client to be able to specify what the best match looks like in these cases. In some cases the lower value is better while in other cases the higher value is better. The best choice is the result of optimizing a function over the range or set of values allowed by the client. *Semantic matching* as discussed in (Paolucci et al., 2002) is an inexact method for matching services based on their semantics, and holds much promise.

In an ad hoc networking environment, clients must be more flexible in requesting and using services. Allowing flexible or inexact matching is extremely important in an environment where all interactions are opportunistic and many of the participants may be resource poor. While an inexact matching policy seems very important in ad hoc networks, it comes at a cost. Exact matching can return the first match and terminate. Inexact matching needs not only to verify the relationship between the advertised value and the acceptable range, but it also needs to compare each match against all other available matches. Simply discovering a match doesn't mean that another advertisement, not yet considered, couldn't yield a better match. These enhancements to the requests formulated by clients and to the matching algorithms may lead to costly searches. In ad hoc networks this can be critical. While in a wired setting a client can afford to browse a service repository, in ad hoc settings the connectivity may not be guaranteed for such a long period of time. This constraint affects the type of search a client performs. A synchronous search usually assumes reliable connectivity, but for an ad hoc networking environment an asynchronous approach can be better for several reasons. The client delegates the effort to perform the search to another party (possibly a searching service) and, if possible, continues execution while waiting for a notification. This approach abstracts issues of consistency of the data in the service registry away from the client. The result of a match is usually a handle to a service. Returning multiple handles forces the client to filter them locally and to choose the best fit. In ad hoc networks with low power hosts this can become a problem as discovering multiple services and filtering them can be resource intensive. Hence, matching algorithms designed for ad hoc networks must be designed to operate in a distributed fashion, so that the filtering occurs on the provider side rather than the client side.

When a service handle is returned, it is paired with a service identifier. This can be

used by the client to access the service in the future, provided it is within communication range. The service provisioning framework ensures that the ID is globally unique. Beside the service ID, the client can receive the provider's profile, a URI to the provider's profile, or a proxy object that represents the provider locally to the client. The simpler the return type, the more complex the processing on the client side. This requires more sophisticated clients which are able to filter the results themselves, learn communication protocols, and so forth. The advantage is reduced bandwidth consumption and framework simplification at the expense of more complex clients. An alternate solution is the proxy object approach. At the expense of higher bandwidth usage, the client retrieves a proxy object that abstracts details of communication with the provider. When it is no longer needed, the proxy object can be discarded, thus saving memory space. The proxy idea is implemented in Jini (Edwards, 1999). Jini uses RMI (Sun-Microsystems, 2003b) to download the proxy object locally to the client and then to mediate the interaction between the proxy and its server, even though they may communicate without using RMI (Sun-Microsystems, 2003b). The proxy object approach is more elegant but it brings its own challenges, e.g., it implies a mechanism for code mobility as suggested in μCODE (Picco, 1998). From the ad hoc perspective, the tradeoff between bandwidth and power is a challenge as both commodities are in short supply in the ad hoc setting. In general, solutions differ appropriately from system to system.

DISTILLED IMPERATIVES FOR DISCOVERY IN AD HOC SETTINGS
   • The request should be comprehensive to enable semantic matching which can maximize hits, which is useful in environments with transient interactions and decoupled computing.
   • The matching algorithm should be flexible and should return the fewest possible results to reduce client side processing.
   • The handle to the service should have a small footprint but should be able to abstract details of communication and coordination in ad hoc networks away from the user of the service.

Invocation

The next step after discovering a service is to use or invoke it. This step is one of the most challenging in the ad hoc wireless setting. Consider the scenario in our example where Robert is downloading soil analysis data from the soil analysis service. When the download is half completed, Robert unknowingly walks out of range of David's PDA. This breaks the connection and terminates the download. This scenario is typical of an ad hoc network where frequent and unannounced disconnections abound.

The biggest challenge is to maintain connectivity for a sufficiently long period of time. If we could assume that the connectivity lasts for the entire duration of the service discovery and invocation, the dynamic character of an ad hoc network would be reduced to that of a reliable wired network. A solution which mitigates this problem is the notion of context-sensitive bindings (Sen & Roman, 2003). Context sensitive bindings create a binding between the user and provider of the service and try to maintain it for the duration

of some interaction between them. If the provider goes out of range, the framework tries automatically to redirect the user to another similar provider or else to fail gracefully. In addition to maintaining the binding between the client and the service, context-sensitive binding mitigates the effect of frequent disconnections by periodically caching the state of the interaction between the client and the service. Context-sensitive bindings can also use the notion of a *motion profile* published by a service provider in its advertisement to estimate disconnections before they occur and to judge how long the disconnection will last. If an imminent disconnection seems to be permanent, a service invocation may be cancelled. However, if the analysis of the motion profile indicates possible future contact between the two parties, an asynchronous type of interaction can be employed. The client only needs connectivity initially until the method execution is launched, that is until the parameters of the call are shipped and the call is made on the remote host. The connection between the two ends can be dropped after that, until some result is ready for the client. Some notification mechanism must tell the client that there is a result ready or the client will need to poll for the result when the connection is restored. A more sophisticated version of the above example can entail result delivery to a remote location. It is possible to deduce from motion profiles that the client moves towards a certain destination while its task is being processed. Geocast (Ko & Vaidya, 1998) protocols can help deliver the result in a certain physical area while Mobicast (Huang, Lu, & Roman, 2003) can help coordinate both the time and the place of result delivery.

Context sensitive binding can be realized in the form of middleware that supports the functionality. We developed a basic middleware to demonstrate proof of that concept. A programmer can specify an interface (analogous to a service request) that he may be interested in. The middleware then searches a repository for providers that fulfil all the needs specified by the programmer. A connection is then created between the client and the provider and maintained for as long as there is wireless connectivity between them. In the event of a disconnection, the middleware automatically looks for an alternate candidate and rebinds to it in a manner transparent to the application on the client side. Periodic images of the state of the interaction are cached on the client so that when there is a switch between providers, the cached state can be pushed to the new provider to ensure that the interaction does not have to revert to the first step.

Once the provision for a connection is made, the next consideration is the protocol that is to be used. The simplest discovery mechanisms use well known remote procedure call protocols like RMI (Sun-Microsystems, 2003b) or SOAP (XML-Protocol-Working-Group, 2003). At the next level lie services that can *learn* the communication protocol described in the advertised service profile. The clients need to share an ontology with the provider offering the services, to be sure they understand the description of the protocol. WSDL (W3C-XML-Activity-On-XML-Protocols, 2003) is an example of an XML-based language for describing services and how to access them. The last category consists of models in which clients do not need to know anything about the communication protocol. They use proxy objects as local representations of services. These proxy objects may be sophisticated enough to implement the entire service themselves, or they may be client side handles to the server providing the service. Each proxy is used by the client to interact with the

actual implementation of the service. The proxy handles the communication with the provider and the client only needs to have knowledge of the interface the proxy offers. We observe again that the client and the provider's proxy need to adhere to the same ontology. The service provider needs to address issues like the communication protocol, how to handle disconnections, and dynamic rebinding of the proxy to another instance of the service running on a different machine while the framework offers support for downloading code and possibly introspection. An example of an implementation of this model is Jini (Edwards, 1999). Support for proxy downloading and remote method invocation is ensured by RMI (Sun-Microsystems, 2003b), while the client is expected to know how to interact with the proxy object (e.g., it may use a given Java interface which the proxy object implements). We stress that Jini represents one possible implementation of this model. Alternate implementations of over-the-air-downloading of software exist, the m-services (Paurobally, Turner, & Jennings, 2003) architecture being a good example.

DISTILLED IMPERATIVES FOR INVOCATION IN AD HOC SETTINGS
- Provide a mechanism that mitigates the problem of frequent and unannounced disconnections.
- Use motion profiles to predict where a certain host will be at a given time and plan interactions using this information.
- Have a sophisticated proxy that can hide communication protocol details from the user and perform tasks like rebinding to better service providers without user intervention.

Composition

Service composition is the notion of taking two or more autonomous services and combining them to behave like one composite service. We illustrate the usefulness of this concept by referring back to our original example. Recall that Robert's task was to generate a map of soil pollutants. To achieve this, Robert initially requested a service which gave him soil analysis data. However, as he started receiving information from the service, he realized that he would require location data to generate a map. The soil analysis service did not have location data so he went to a GPS location service to find the relevant data. Even with location information, there is not enough data to build a map. Hence, Robert *composed* the soil analysis service and the location service with the mapping service to generate a map with all the relevant data. Service composition is a powerful concept that allows multiple stand-alone services, each of which is highly specialized to a particular task, to be combined into a single service which can then offer an interlocked collection of services under a single umbrella. The benefit of the composed service is generally greater than the sum of benefits yielded by its constituents.

Composition promotes separation of concerns in the development of services. In our example, if the soil analysis service had to develop its own location generating program, it would most likely be inferior to that of a dedicated GPS location service. By composing itself with a third party location service, it effectively "outsources" the problem of gathering location data, saving the cost of development of a location data gathering tool while at the same time providing a better quality of service. Composed services can differ based on the

way they are constructed and used. We first discuss ways of building a composed service, followed by ways a composed service may be used.

The two main approaches for composing services are the distributed approach and the integrated approach. In the distributed approach, the composed service behaves like a distributed application with component services residing on different machines. The composing mechanism may use the Façade design pattern (Gamma, Helm, Johnson, & Vlissides, 1994) to provide a single, common interface to the user (Mennie & Pagurek, 2000). The user makes local method calls to the interface which determines which of the component services can handle the request and uses a protocol such as RMI (Sun-Microsystems, 2003b) to call that service. Results are returned to the interface which disburses them to the user locally. In the integrated approach, the code for all the services is aggregated by the composing entity and runs locally. Regardless of the approach for composing a service, it is critical to ensure that the elements of the composed services neither conflict with each other nor cause the composed service to deadlock. In essence, if we have n services that have been successfully composed, adding the $n + 1^{st}$ service should not violate the integrity of the n original services (Kirner, 2002). This can be verified in some cases through formal reasoning.

Composed services can be used in two ways. In automatic composition, the user makes some request to a specific service, e.g., Robert requests a table of locations and soil makeup from the soil analysis service. The soil analysis service checks its data and discovers that it can service the request for the soil analysis but it cannot provide location information. Hence, to fulfil the request, it automatically requests that a location service provide location data on its behalf. In object oriented terms, this process is analogous to a method call on an external object. Notice that to Robert, all the data seems to be coming from a single source, the soil analysis service. In manual composition, the user manually selects a list of services that he or she believes will yield a useful result when combined. These services are then integrated according to the specific request. They do not seek out additional services to help them complete their tasks if they are lacking in some capability. The onus is on the user to ensure that such a situation does not arise.

We now discuss the merits of each approach as it pertains to ad hoc networks. The distributed approach to composing services is less reliable in ad hoc networks because a single element moving out of range can affect the entire composed service whereas if the services are hoarded, the connectivity needs to be preserved for a shorter period of time during which the services are be copied over to the composing host. However, the downside of hoarding services is that everything needs to be run locally which can drain power or tie up crucial memory. Another scenario is that where a certain resource is available only on the provider host, e.g., a specialized sensor. The best compromise in such situations is to exploit the notion of motion profiles as mentioned in the previous subsection, hoard only those services which are likely to move out of range, and use the stable proximal ones in a distributed fashion.

For composed services, the automatic composition approach is more user-friendly but it does raise a concern that affects all composed services, that of security and confidentiality.

Issues of security and privacy come to the fore in automatic service composition, especially if there is an exchange of personal information, e.g., Robert might be willing to share authentication information with the soil analysis service since he knows it is reliable and has appropriate security measures to safeguard his information. However, he may not want his information transmitted to the location service since he does not trust its security system. Such an event could lead to loss of privacy without the individual being aware of it. With manual composition, one can presume that the user would select only those services that he or she trusted.

Service composition is critical in the ad hoc wireless setting for two reasons. An ad hoc network by definition, is an "anytime, anywhere" network. This means that an ad hoc network encounters a dynamic environment with its own nuances, requiring customized, situation specific services. Having large, indivisible services which cater to specific needs is not flexible enough for such dynamic environments. It is much more effective to have a set of smaller services and select a subset of them to form a customized service on demand. Composition supports such a capability. The second reason why composition is an essential feature of ad hoc networks is that it allows for highly specialized services with potentially smaller footprints. This means that such services can run on small devices like PDAs or cellphones, increasing the number of devices in the network that can potentially be service providers, and thereby increasing the number of basic and composed services available in the network.

The final issue we discuss is related to the control of a composed service with an eye toward security issues. Note that the elements of a composed service are provided by many different providers. Also observe that to use a provided service in a composition, the composing entity has to obtain some degree of control over the provided service. The provider may not necessarily want to relinquish control over its service and hence a middle ground must be found. We envision the use of management software like Java Management Extension (Sun-Microsystems, 2003a), which can manage services from different providers and yet allow them to retain some degree of autonomy.

DISTILLED IMPERATIVES FOR COMPOSITION IN AD HOC SETTINGS
• The approach to composing services should consider a hybrid strategy of integrated and distributed composition, which exploits the state of the network and services to the fullest.
• Composable services should be designed so as not to conflict with other services, to increase the range of allowable compositions.
• Security and management systems should be compatible to maximize interoperability.

Future Trends

While service oriented computing has reached a certain level of maturity in wired networks, it remains a nascent area of research in ad hoc wireless networks. Much potential for future research exists. In this section, we briefly describe some of the key

research issues which we feel hold technical challenges and/or promise beneficial impact. Service descriptions can be enhanced to include motion profiles, battery power levels and composability attributes. The content of a description could also be made dynamic to incorporate the *current* status of a service, e.g., a printer would reliably advertise how much paper is remaining in its trays. In the area of service advertisements, one can envision active advertisements where services do not wait for a request but actively solicit jobs. The idea of a client advertising jobs rather than providers advertising services is also one that could be explored. Providers would examine a pool of jobs and pick up ones that they could perform. A provider would send a notification to the client when the job was completed. Another novel idea is time limited advertisements, analogous to limited time discount offers. A provider could offer lower cost or higher quality services at certain times when the load on the network is low and the service is not being fully exploited.

In the arena of discovery, approximate matching functions based on semantics is an open area of investigation. These algorithms could be tailored to ad hoc networks, with decision making based on the available content and urgency of requests. This direction of study could extend work on metrics for service discovery and usage - functions that help a user decide which is the better of two or more given services. Service invocation also poses some interesting problems, e.g., the notion of "follow-me" services is one in which the client holds the service proxy but the providing host changes over time due to physical mobility. This work can be related to resource leasing - pushing services onto resource rich hosts and controlling them through proxy objects and delivering results remotely, e.g., a client requests a job in St. Louis, shuts down, restarts in New York and receives the result from a local access point. The concept of context sensitive binding can be applied as well, to facilitate active resource monitoring and load balancing based on network traffic.

The concept of composing multiple instances of a service for parallel processing and performance gains is extremely useful in ad hoc networks when the connection may exist for short periods of time only. This would require specification of dependencies between services with a hierarchy ranging from required to preferred. It also raises issues of combining code from various sources seamlessly and ensuring that the result exhibits correct and expected behavior.

Other than work related to enhancing the *elements* of service oriented computing, there are some broader trends that can improve service oriented computing as a whole. One such idea is the use of virtual currency, a version of which has been proposed in (Buttyan & Hubaux, 2001). Another concept is that of a language independent representation of services, where the skeleton of the description highlights the semantics of the service and appropriate words from any language can be used to describe those semantics. This would allow the framework to understand requests in any language as long as the semantics of the service remained consistent. Lastly, one cannot ignore the security implications of such a framework which will require much research into online security, especially in scenarios where two programs can interact with each other without human intervention.

Conclusion

We began this chapter by introducing the notion of service oriented computing and its applicability in ad hoc wireless networks. We discussed existing service oriented computing models and highlighted the *Semantic Web* as an example of service oriented computing at work. We then discussed issues of service oriented computing in ad hoc wireless networks as they pertain to each of the salient elements of a service oriented computing framework, viz. description, advertisement, discovery, invocation and composition. For each of these elements, we distilled a set of imperatives for the ad hoc wireless setting. Finally, we highlighted key research issues and enhancements which we see as defining the next steps for research in this area. We hope this chapter has given a perspective on the issues, challenges and imperatives of designing and implementing a service oriented computing framework for ad hoc wireless settings, one that will serve as a guide for future research.

Acknowledgements

References

Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., & Sycara, K. (2002). Daml-s: Web service description for the semantic web. In *Proceedings of the 1st international semantic webconference.*

Berners-Lee, T., Hendler, J., & Lassila, O. (2001, May). *The semantic web.* Scientific American.

Buttyan, L., & Hubaux, J. (2001). *Nuglets: a virtual currency to stimulate cooperation in self-organized ad hoc networks* (Tech. Rep.). EPFL.

Cohen, J., & Aggarwal, S. (1998, July). *General event notification architecture.* http://www.globecom.net/ietf/draft/draft-cohen-gena-p-base-01.html.

Edwards, W. K. (1999). *Core jini.* Sun Microsystems Press.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999, June). *Request for comments 2616 - hypertext transfer protocol - http/1.1.* ftp://ftp.isi.edu/in-notes/rfc2616.txt.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns - elements of reusable object-oriented software.* Addison Wesley.

Gerlenter, D. (1985, January). Generative communication in linda. *ACM Computing Surveys, 7,* 80 - 112.

Goland, Y., Cai, T., Leach, P., & Gu, Y. (1998, April). *Simple service discovery protocol.* http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt.

Handorean, R., & Gill, C. (2004). Accomodating transient connectivity in ad hoc and mobile settings. In *To appear in proceedings of pervasive 2004.*

Handorean, R., & Roman, G.-C. (2002, April). Service provision in ad hoc networks. In *Proc. of the 5*[th] *international conference on coordination models* (p. 207-219).

Horrocks, I. (2002). Daml+oil: A description logic for the semantic web. *IEEE Bulletin of the Technical Committee on Data Engineering.*

Huang, Q., Lu, C., & Roman, G.-C. (2003, April). Mobicast: Just-in-time multicast for sensor networks under spatiotemporal constraints. In *Lecture notes in computer science.* Springer-Verlag.

Kempf, J., & Pierre, P. S. (1999). *Service location protocol for enterpriser networks: Implementing and deploying a dynamic service resource finder.* John Wiley and Sons.

Kirner, R. (2002, October). Enforcing composability for ubiquitous computing systems. In *Proceedings of the 7th cabernet radicals workshop.*

Ko, Y., & Vaidya, N. (1998). *Geocasting in mobile ad hoc networks.* Location-based multicast algorithms.

Mennie, D., & Pagurek, B. (2000, June). An architecture to support dynamic composition of service components. In *Proceedings of the 5th international workshop on component -oriented programming (wcop 2000).*

Microsoft-Corporation. (2000, June). *Universal plug and play device architecture.* http://www.upnp.org/download/UPnPDA10_20000613.htm.

Miller, S. (2003, October). *The autoip publisher page.* http://www.autoip.net.

Murphy, A., Picco, G., & Roman, G.-C. (2001, April). Lime: A middleware for physical and logical mobility. In *Proc. of the 21st int'l conf. on distributed computing systems* (pp. 524–533).

Paolucci, M., Kawmura, T., Payne, T., & Sycara, K. (2002). Semantic matching of web services capabilities. In *Proceedings of the 1st international semantic web conference.*

Paurobally, S., Turner, P. J., & Jennings, N. R. (2003). Towards automating negotation for m-services. In *Proceedings of the 5th international workshop on agent-mediated e-commerce* (p. 124-131).

Picco, G.-P. (1998, September). code: A lightweight and flexible mobile code toolkit. In *Lecture notes on computer science* (Vol. 1477, p. 160-171). Springer.

Salutation-Consortium. (2003, October). *The salutation consortium homepage.* http://www.salutation.org.

Sen, R., & Roman, G.-C. (2003). *Context-sensitive binding: Flexible programming using transparent context maintenance* (Tech. Rep. No. WU-CSE-2003-72). Washington University Department of Computer Science.

Srinivas, J. (1995, August). *Rfc 1831 - open network computing remote procedure call protocol specification.* http://www.ietf.org/rfc/rfc1831.txt.

Storey, M., Blair, G., & Friday, A. (2002). Mare - resource discovery and configuration in ad hoc networks. *Mobile Networks and Applications, 7,* 277-287.

Sun-Microsystems. (2003a, October). *Java management extensions homepage.* http://java.sun.com/products/JavaManagement/.

Sun-Microsystems. (2003b, October). *Java remote method invocation page.* http://java.sun.com/products/jdk/rmi/.

UDDI-Organization. (2000). *Uddi technical white paper.* http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf.

W3C-Metadata-Activity. (2000, March). *Resource description framework schema specification 1.0.* http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

W3C-Semantic-Web-Activity. (2003, October). *Worldwide web consortium page on resource description framework.* http://www.w3.org/RDF/.

W3C-XML-Activity-On-XML-Protocols. (2003, October). *W3c recommendation: Web services description language 1.1.* http://www.w3.org/TR/wsdl.

XML-Core-Working-Group. (2000, October). *W3c recommendation: Xml version 1.0 second edition.* http://www.w3.org/TR/2000/REC-xml-20001006.

XML-Protocol-Working-Group. (2003, June). *W3c recommendation: Soap version 1.2 parts 0-2.* http://www.w3.org/TR/SOAP/.