Contexte et modélisation de l'Environnement d'Exécution Dynamique d'une application logicielle

Context and modelization of the Dynamic Runtime Environment of a software application

Author :
Nicolas Bussière (Master RSD, EPU)

Supervisors :
Stéphane Lavirotte (I3S, IUFM de Nice)
Éric Mathieu (MobileGov)
Jean-Yves Tigli (I3S, EPU)

Abstract

Ubiquitous computing appeared at Xerox PARC twenty years ago [28]. Originally, applications were done in an ad-hoc style. Things evolved with the apparition of Service-Oriented Architecture and Web Services. Dealing with devices and the implied eventing mechanisms added improvements but also some associated difficulties. Number of direct interactions can be reduced by introducing context. We will present mechanisms providing contextual-filtering for both service discovery and during the following communications.

Acknowledgements

I would like to thank the Rainbow team for having welcomed me and particularly those who shared the Ubiquarium with me. I address special thanks to Jean-Yves and Stéphane for having taught me more than a way of working and for their presence and help especially in some difficult moments.

Contents

List of Figures

Chapter 1

Introduction

We will interest in this document to service-based applications in a **ubiquitous** (potentially **mobile**) **computing world**. Such application are driven by users (in its most generic definition as a user can potentially be another application in case of **device-to-device interactions**). Each application is running on a given **service infrastructure** and has to adapt to the **constraints** imposed by this **environment**. In a mobile world, disconnection can be frequent and service availability is not a guarantee. To increase **robustness** and to facilitate **service's continuity**, each basic functional capability thus need to be provided by more than one unique provider. This induce another problem, if several services are available to perform a task, the most **relevant** solution should be selected.

1.1 From Services to Web-Services for Device

After having recalled the principles of **Service-Oriented Architecture** and **Web Services**, we will explain how these approaches can be helpful to deal with **devices** and even **heterogeneous devices**.

1.1.1 Service-Oriented Architecture (SOA)

Service-Oriented Architectures (**SOA**) have been created to encourage code re-use and make **A**pplicationapplications easier to adapt. This is done naturally as a computer service is aimed to be a representation of a **business component**. Each service only focuses on a specific task thus encourage **service re-use** each time the same task need to be performed. As it is suggested in the name, **SOA** are based on **services**.

1.1.1.1 Service Characteristics

A **service** is a **software entity** which can be invoked, which has a specific functionality, and a well defined **interface**. **Service** provide an **abstraction**

layer in the **architecture** as objects do in **object-oriented programming paradigm**. **Service granularity** is however much more larger for services than for objects. The **encapsulation principle** give some freedom in implementations: a given **service** can actually be provided by different implementations as far as the **interface** is respected; this gives some freedom in the selection of **service provider**. Every **service** is described in a **description language** fixed by a **service framework**. For better **reutilisability**, **extensibility** and **dynamicity**, **services** are not hard-linked together to create **applications** as it is the case in the **functional** or **object-oriented programming paradigm**; interactions between **services** are specified independently of the functional code and handled by the **service infrastructure**. This is what is called **loose coupling**.

1.1.1.2 Services Collaboration

Services composing a **SOA** need to known at least another **service** to communicate with (either a real service or a **service directory**). Although this can be done statically, **SOA** were designed to enable some **dynamicity** in **service interactions**. In order to let new services join the **architecture** dynamically (at runtime), a **service discovery mechanism** is needed.

SOA have some advantages but it still remains a problem when trying to build an **application** with **services** coming from different **service frameworks**. Incompatibilities have been overcome as we will see in next section by using some **Web Technologies** to accomodate with existing **heterogeneities**. Afterwards we will focus on devices and their associated difficulties: **distributed service discovery** to deal with **device mobility** and **eventing mechanism** increasing **applications' reactivity** to changes in devices' state. Finally, we will expose in section 1.1.4 an approach combining the advantages of both previous ones.

1.1.2 Interoperability of Web Services

To enhance **interoperability** between several services from various **SOA** infrastructure, **Web Services** were created. Not to be limited to a given **programming language**, they only specify standards for data representation format and for distant method invocations. They rely on de-facto standards such as **HyperText Transfert Protocol**[1] (**HTTP**) for network communications and **eXtensible Markup Language**[2] (**XML**) for data representation.

Since **Web Services** are inspired from **SOA**, they are based on description of the provided services: **Web Service Description Language**[3] (**WSDL**). As **Web Services** are accessible across the Web, the **Universal Description Discovery and Integration**[4] (**UDDI**) protocol was created to facilitate **web**

[1] http://tools.ietf.org/html/rfc2616
[2] http://www.w3.org/TR/xml
[3] http://www.w3.org/TR/wsdl
[4] http://www.oasis-open.org/committees/uddi-spec/tcspecs.shtml

service discovery (registered services can be found in a **centralized directory**). Finally method invocation are addressed by **XML Remote Procedure Call**[5] (**XML-RPC**) for simple procedure (stateless) and **Simple Object Access Protocol**[6] (**SOAP**) for interaction with objects.

Today, the **Web Services Interoperability Organization Consortium**[7] (**WS-I**) is coordinating **Web Services** developments. This organization provide guidance, recommended practices and supporting resources as a common base to promote **interoperability**.

1.1.3 Service for Hardware

Services are limited to the computing virtual world. To diversify their functionnalities, the **service architecture** can also be applied on **devices**. As we will see later on, **devices** are between the real world and the virtual computing modelization. Since the real physical world is following its own rules that devices have to adapt to. This induces some difficulties. **Events** are needed to increase **reactivity**. **Discovery protocols** also need to be adapted to find a potentially **mobile device** in the wideness of the world.

1.1.3.1 Device

We call **input/output devices**, or simply **devices**, equipments interacting with their **physical environment**. **Devices** are basically similar to **services**: they provide a functionality, a way to access it, and are **loosely coupled** between each others. However, keyword[Physical constraints]physical constraints, also called keyword[Resource Dependency]resource dependency, make them different. A **device** can appear or vanish at any time. This can be due to real **mobile devices** or to specific **constraints** inherited from the real world such as an **energy saving policy** for example. To increase **reactivity**, **device** can use **asynchronous communication** to notify applications of changes in their state. This **eventing mechanism** provide a **push mechanism** less costly and more efficient than a **polling strategy**.

1.1.3.2 Service-Oriented Architecture for Device (SOAD)

Benefits of **SOA** can be extended to **devices** management. As we have seen, there are some similarities, but points have to be extended. We study here this reunition of **SOA** and **devices** worlds. We will present two representative concepts: **discovery** and **eventing mechanisms**.

1.1.3.3 Discovery and Publication

Discovery and **publication** both helps the bootstraping of any **mobile application**. Without this initial step, no dynamicity would be possible. This

[5] http://www.xmlrpc.com
[6] http://www.w3.org/TR/soap
[7] http://www.ws-i.org

would implied that devices and services must be statically known for any application to be able to run. **Publication** is done by service producer, they inform others of their available services. They can either register to a central server (yellow pages approach) or advertise themselves by broadcasting periodically on the network. On the other hand, discovery is done by service consumer which proactively searches for available services. Contrarily to **SOA** for which **service discovery** was most of the time assumed by a **central service directory**, **SOAD** prefers **decentralized discovery mechanisms** not to be dependent on any **infrastructure**.

Liscano distinguish 3 kinds of discovery mechanism [17]. Firstly, he present the **Distributed Pull Service Discovery**. The main part of the system is based on service request broadcast messages (step 1 in figure 1.1). Only **services provider** compliant to the request answer to the service consumer (step 2) after what normal communication can take place (step 3).

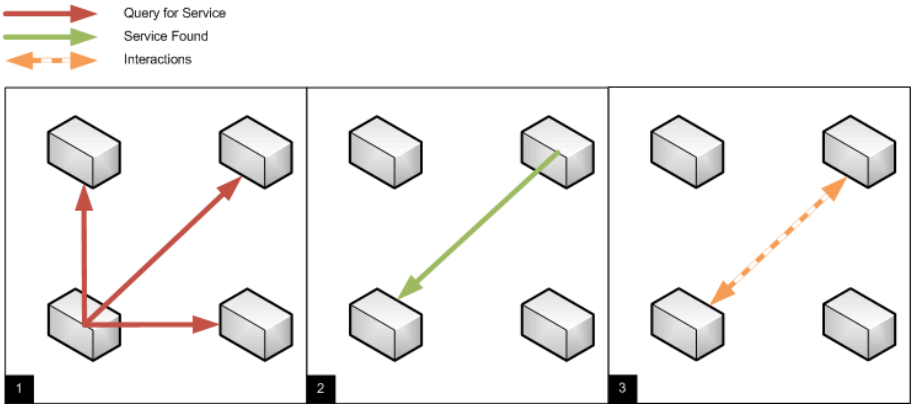

Figure 1.1: **Distributed Pull Service Discovery**

This first approach highly rely on a **broadcast-enabled network**. To deal with networks for which broadcast is too expensive and to be less dependent to technology, the second approach depicted (**Centralized Pull Service Discovery**) is based on a central mechanism. As the previous one, information are pulled by the **consumer** when needed. Contrarilly to the distributed version, queries are addressed to a **central repository** (figure 1.2 step 1). An example of such centralized mechanism is the **service directory** in **SOA**. The central repository can either reemit the query but it can also answer directelly if it already have the information in its local cache. Caching strategy imply that queries are periodically sent to build a list of available services and moreother to keep it up to date. If fulfilling queries, services reply to the **central repository** (step 2) rather than to the query initiator. The central place then forward the answer to the consumer (step 3) and no more interactions are needed with

the central repository (step 4).

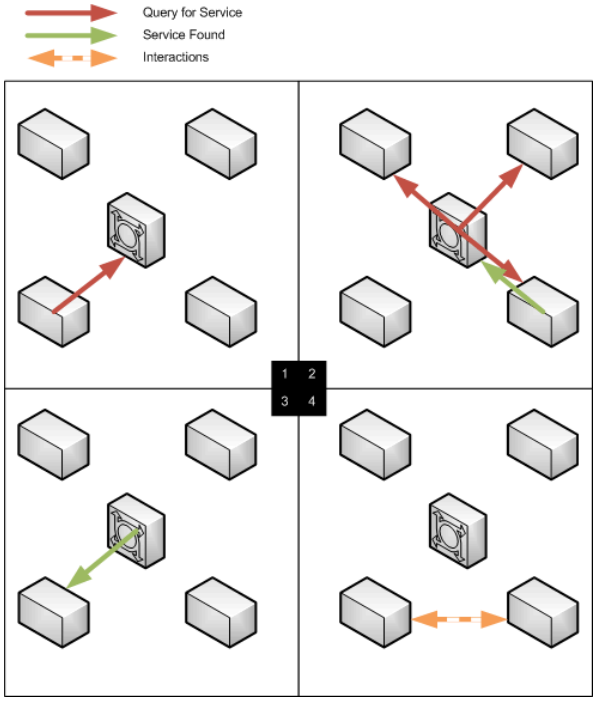

Figure 1.2: **Centralized Pull Service Discovery**

Last architecture explained by Liscano is the **Centralized Push Service Discovery**. As the previous one it is centralized, thus not relying on broadcast-capability of the underneath network. In the **centralized pull mechanism**, the central repository still need to use some kind of broadcasting to discover services; here the **push mechanism** really prevent from doing any broadcasting. Here, each **service provider** must advertize itself to the repository (first step in figure 1.3). For the same coherence reason as previously, this has to be done periodically so that **services** which are no more available will not be said available by the **repository** if they do not exist any more. If the system is fully based on **push mechanism**, **services** are advertized to **service consumer** (step 2); if not, **service consumer** address same queries as in the **centralized pull case**. As in all other cases, once discovered, communication are done directly between **producer** and **consumer**.

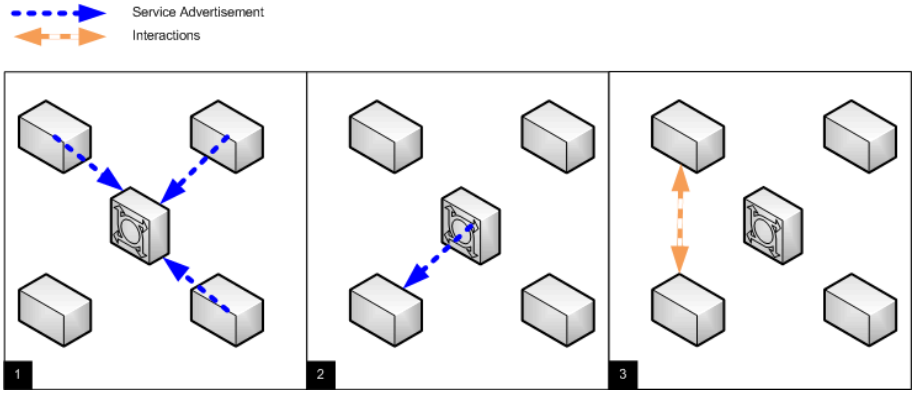

Figure 1.3: **Centralized Push Service Discovery**

Finally, we also have **Distributed Push Service Discovery**. Although not presented by Liscano, it is used for example in network routing with the **Border Gateway Protocol (BGP)**. The principle is derived from the **centralized pull mechanism** without any centralization. The central point can be circumvented if service produce have the possibility to broadcast their advertizement to all potential **consumer** (figure 1.4, step 1). Each consumer is than in charge to store received advertisement or to wait until a new advertisement for a desired producer.

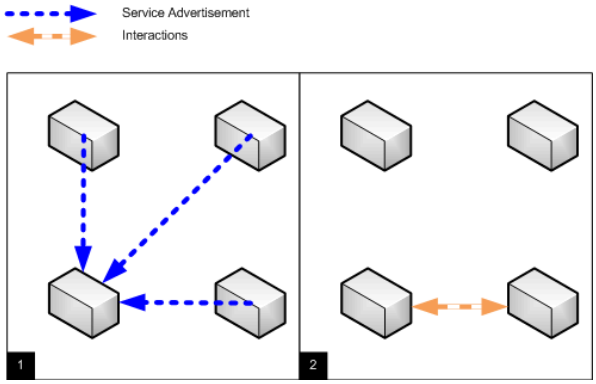

Figure 1.4: **Distributed Push Service Discovery**

Decentralized systems are best fitted for **devices** (even more for **mobile devices**) which are dynamic by nature. Adaptation to this **dynamicity** also contribute to the **robustness** of the whole infrastructure.

As introduced in section 1.1.3.2, we will now explain how **devices** can in-

crease their **reactivity** with an **eventing mechanism**.

1.1.3.4 Eventing Mechanism to Increase Reactivity

Synchronous communication mechanism are well fitted for **services**-based **applications**. **Devices** do not only provide functions, they may also need to advertise from a modification of their state. Consequently, **communications** need to occur in both ways, and may happen when the **device** needs it, without an external request. This is a crucial point for the **reactivity** of the system.

The **asynchronous messaging** mechanism used by **services for devices** is **event** notification. Consumers interested in receiving events from a **device** have to **subscribe** to them. **Services for devices** thus have to handle **subscriptions** along with **event** sending. **Events** generally notify changes from the **physical constraints** of a **device** (for example a battery level, a sensor information change) or for **interaction devices**, a change in their state (for example a pressed switch).

1.1.4 Interoperability of Devices in a Service Oriented Architecture

Web Service suffer from the same problem than **SOA** when trying to interact with **devices**. **Web Services** provide mechanism essentially for **synchronous communications**, that is why **Web Service for Devices** (**WSD**) were introduced. Although keeping the **interoperability** advantage of the **Web Service** approach, **WSD** also take into account the **eventing mechanism** introduced earlier (section 1.1.3.2).

The **WSD model** (See figure 1.5) has two major implementations: **Universal Plug and Play**[8] (**UPnP**) and **Device Profile for Web Services**[9] (**DPWS**) that we will describe in more details in following sections.

1.1.4.1 Universal Plug and Play (UPnP)

UPnP was firstly created to extend the **Plug and Play** concept to peripherals on a **network**. It defines **device profiles**, in a specific language (though **XML**), for example for printers, or gateways, which are fixed by the **UPnP Consortium**, and implemented by **UPnP servers** in such lightweight **devices**. The **Simple Service Discovery Protocol**[10] (**SSDP**) used by **UPnP** permits to **discover** and search **devices** on a **local IP network**, using **multicast UDP messages**. It can moreover specify a **type filter** for searches, and then find only **devices** or **services** matching this **type**. **UPnP** uses **Simple Object Access Protocol SOAP** for classical **Web Services** requests. The **eventing layer** is managed by the **General Event Notification Architecture**[11] (**GENA**). Subscription to **events** is required, and leased. However, this

[8] http://www.upnp.org
[9] http://specs.xmlsoap.org/ws/2006/02/devprof/devicesprofile.pdf
[10] http://www.upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf
[11] http://msdn2.microsoft.com/en-US/library/Aa505982.aspx

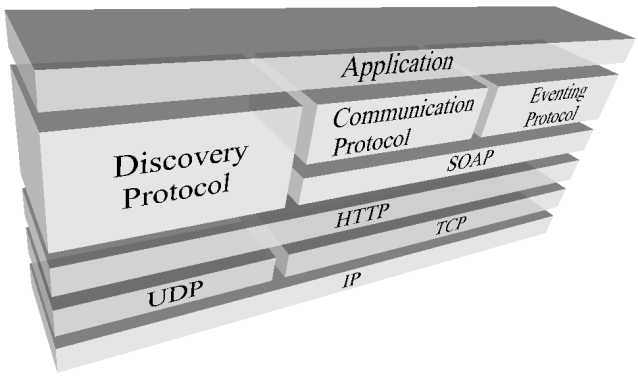

Figure 1.5: **WSD Stack**

protocol has some drawbacks: **events** are related to **state variables**, and must be defined before use. When a **service** contains several evented variables, all their values are sent at once even if only one value has changed.

1.1.4.2 Device Profile for Web Service (DPWS)

Due to the limitations previously exposed for **UPnP**, and to the non-standard descriptions and protocols used, the **DPWS** technology has emerged with an aim of replacing **UPnP**. **DPWS** was created more as a **Web Service** than as a **plug and play** protocol. It is based on **Web Services** standards, such as **SOAP** for data transfers and **WSDL** for descriptions. This **Web Services** kernel is extended by some WS-* specifications: **WS-Discovery**[12] and **SOAP-over-UDP** for **peer-to-peer service discovery** at a **local network** scale, **WS-Eventing**[13] for managing **subscriptions** for event channels, **WS-Security**[14] for **security**, which was cruelly missing with **UPnP**, **WS-Addressing**[15] for advanced endpoint and message addressing, **WS-Policy**[16] for **policy rules** and exchanges, **WS-Transfer**[17] and **WS-Metadataexchange**[18] for **device** and **Service description**.

1.1.5 Conclusion on Services

Services and more generally **SOA** have proven their utility in software engineering. The mechanism was developped in two directions to add **interoperability** on one hand (**WS**) and **device handling capability** on the other

[12] http://specs.xmlsoap.org/ws/2005/04/discovery/ws-discovery.pdf
[13] http://www.w3.org/Submission/WS-Eventing
[14] http://www-128.ibm.com/developerworks/library/specification/ws-secure
[15] http://www.w3.org/Submission/ws-addressing
[16] http://www.w3.org/Submission/WS-Policy
[17] http://www.w3.org/Submission/WS-Transfer
[18] http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf

		Device handling	
		software	device
Inter-Operability	Legacy	**SOA**	**SOAD**
	Interoperable	**WS**	**WSD**

Table 1.1: 2-dimentionnal Evolution from **SOA** to **WSD**

(**SOAD**).**WSD** were introduced to keep the advantages of both previous improvements (as defined in table 1.1).

As we can already see in our everyday life, the number of **communicating devices** is constantly increasing. One day or another, this could lead to a problematic situation. In order to limit some of the unnecessary **communications** between **services**, a **filtering mechanism** need to be used to avoid wasting **network resources** and **computation time** when possible.

1.2 Motivation: Relevance of Services

We had underlined in the previous section that the number of devices was constantly increasing. This motivate a **filtering mechanism** to only interact with service considered **relevant** in regards to a specific application.

1.2.1 Set of Accessible Services

Most of actual **filtering mechanism** are based on technology. With **WSD**, **devices' communications** are limited to the **local network** on which they are. With **mobile devices**, a natural filter is the communication range. These solutions are not easy adapt to software filtering needs as we can define more complex applicative filters than just proximity. Since we are extending the **SOA**, **context** should also be loosely coupled to business and technical code. To be **re-usable**, the context filtering mechanism should not rely on a specific technology.

In **UPnP** for example, all communications are isolated on the **local network**[19]. We can thus filter communications based on some network appartenance. However, this solution is technology-dependent and do not facilitate dynamic evolution of the filters conditions (network address are quite static).

1.2.2 Set of "tagged" Services

As Rey noticed [25], insulated data does not represent a significant value. In fact each data needs an overheading **metadata** to describe it. Some **metadata** concerns **sensors functional capabilities** (resolution, latency, sampling rate, stability, range, covered area, size, autonomy, orientability, calibration data or

[19]This can be viewed as a negative point, but the **local network** can be virtually spaned on multiple site with bridges and **Virtual Private Networks** (**VPN**) solutions

lifetime); other concerns sensed data (precision, quality or stability). In a multi sensor system, such **metadata** are useful when aggregating multiple data or to prune low quality data if another highest quality data is available from another sensor.

Semantic significations associated to services have been initially specified in centralized directories. In SOA, some systems can provide various details on services. For example in **Geographic Information System (GIS)**, services are annotated with geographical metada [9]. These approaches are also sometimes compared to phone books. **White pages** are used for simple information searches; **yellow pages** enables searches based on functionnalities; **green pages** provides more technical informations and **blue pages** [27] gives more freedom and semantic search capabilities.

Since each application may consider things differently, the **filtering mechanism** should be configurable at the application level and should not rely on a specific technology. **DPWS** and **UPnP** provide some simple mechanisms to filter devices (of a given "type" and/or in a given "scope") at discovery time.

1.2.3 Set of Contextual Relevant Services

Most of actual technology enable filtering mechanism that are evaluated only once at discovery time. Since devices' state are dynamically evolving (undergoing changes in the environment), the filering condition should checked each time the concern device's state changes (or at least periodically). This new kind of filtering mechanism should determine the communication upholding depending on contextual conditions. Context will be explained in more detail in next section. We will then present how contextual filtering mechanism can be implemented in section 3

Chapter 2

Context Awareness for Services

Anthropologist Edward T. Hall [10] introduce the term "**high context culture**" to qualify non-spoken mean of communication, such as historical or social well-known facts. **Context** is here a hidden, implicit channel of communication. We can view context as all information perceived without having been explicitly requested.

Etymologically, context comes from the Latin contextus[1]. Literally, it refers to the surrounding part of an expression helping in ambiguity resolution. As an illustration, in mathematics, an horizontal line over a symbol may be a fraction symbol; it may be part of a complex symbol (a square root, a vector arrow ...); it can also be an underlining or a framing mark. Context is answering those questions (Figure 2.1). As human can do, computers are trying to infer such context in **Optical Character Recognition** (**OCR**) software [15] or more generally in any lexico-syntactical analyser.

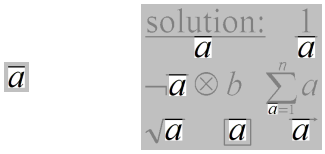

Figure 2.1: **Literal context**: a sample glyph (left) can have several meanings (right)

[1] past participle of contextere: to weave together

2.1 Context Classification

Schilit, Adams and Want [26] categorized context among three axes. Firstly they talk about the physical context. Among all contextual data, we have position, lighting, temperature, noise level. Then they underline the user context. It includes social consideration such as task at hand or people nearby. Finally they evoke the computing context. That gathers all computer dependent aspects (network connectivity, bandwidth, cost, nearby resources, screen size ...). It is also known as the **computing context** or as **resource-aware computing**. It induces constraints linked to the infrastructure and the availble **physical resources** that an **application** must adapt to. They summarize their approach with three questions: Where are you ? Who are you with ? and What resources are nearby ? In [3], Chen and Kotz added time to previous context definition. They also distinguish active an passive context.

2.1.1 Interaction Context

According to Dey [7], "*context is any information that can be used to characterize the **situation** of an **entity**. An **entity** is a person, place, or object that is considered **relevant** to the interaction between a user and an application, including the user and application themselves*". In the **Human-Computer Interaction** (**HCI**) field, Coutaz, Crowley, Dobson and Garlan handle context in a four levels architecture [5] (Figure 2.2).

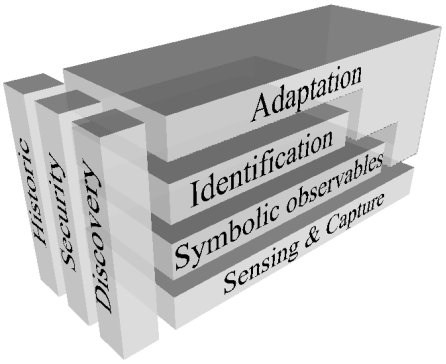

Figure 2.2: Four level architecture and orthogonal services

As in the **Open System Interconnection (OSI) network stack**, the lowest layer (**capture layer**) is handling the **physical data**. These raw **numeric observables** are transformed in the **transformation layer** into **symbolic observables** as in **OSI data link layer** where physical data are forming logical frames. Then comes the **identification layer**. It can be compared to the **presentation layer** of the **OSI stack** as its **role** is to **identify** useful **situations** and to forward them to the upper layer. Decisions are made in the **application**

layer (**OSI**) here called **adaptation layer**. As recent wireless network had outlined some difficulties of the rigid models[2], any layer can short-circuit intermediate layers and have a direct communication with the **adaptation layer**. Three others notions are presented: **historic service**, **resource discovery service** (helping **fault tolerance**) and **security**. Since this architecture does not provide solution to include these important aspects, they are left as **orthogonal services**. In fact they are closely linked together. The historic service can help in fault tolerance. In effect, if communication can be established between a sensor and an application at a given time, the application may consult the sensor historic data afterwards and keep running using a local cache memory and interpolating missing values.

2.1.2 Data Context

Oh, Shin, Jang and Woo present a **unified context** in [21]. Their approach consist of specifying answers to the six questions **Who? Where? What? When? How? and Why?** This is also known as the **5W1H approach**. The first question tends to identify persons who are using the system. The model tries to be exhaustive by including all **personal data** relative to an individual in his identity. Second question deals with **location**. Again to be as complete as possible, all available data are included: a global position and a local position (such as a room relatively to a building). "What" stands for the service asked by the previously identified user. Most of the time it is coded as an **Uniform Resource Identifier** (**URI**). Answering the "When" question will let the system have some temporal references. **Universal time** (hard to synchronize among several **mobile device**) can be used to have a scheduler view of incoming requests. **Relative time** informations may also give usefull information on some **task duration** for example. Answers to the "How" question may include body position or medical data such as blood pressure or cardiac rythm. Finally, the "Why" answers include some user mental aspects' such as intention or emotional state. This last part is difficult to describe and thus to sense.

Since **sensors** are sensible piece of hardware and as they may send erroneous value, some precaution should be taken in order to achieve long-term reliable operation. To be able to sense some value even in the case of hardware failure (**fault tolerance**), **sensors** should be at least doubled[3]. At the other end of the chain, the **consumer application** can only handle a single input; therefore a component is needed in the middle of this architecture to "glue" parts together. Such a component is presented in [6] as a **Fusion Contextor**. The cited document presents a wide diversity of **contextors** to facilitate robust architecture construction as they are capable of estimate their own **Quality of Service** (**QoS**).

[2] In **Wi-Fi** network, when a collision occurs and a packet is lost, standard HTTP mechanisms will ask the source to retransmit the packet thus penalizing all the network, even if the packet was lost between the last router and the destination.

[3] if two **sensors** give different values, a third one might help to determine which of the two is out of order.

2.2 Logical Versus Physical

There are two strategy for handling context. The **logical approach** has a centralized view of the otherwhole system whereas the **physical one** is based on direct interaction between entities without requiring any **global infrastructure**.

2.2.1 The Logical Approach

In his PhD [25], Rey presents a complete logical architecture handling context. The lowest level concept is the **observable** concept. An **observable** is a single data that can be observed. **Observables** can be grouped together to form an **entity**. A **task** may be dependent on an **entity**. Two **tasks** type can be distinguished: **current** and **background tasks**. **Current tasks** (**tasks at hand**) are those on which the user is actually working on. **Background tasks** are **tasks** in a waiting state. An **observable** modification may lead to a switch between the two **tasks** categories. Depending of **entities**, we can infer **situations**. **Situations** are forming **contexts**. Every **situations** of a given **context** have the same set of **roles** and **relations**. A **role** is defining a semantic meaning of an **entity**'s presence in the application. A **relation** explicits the semantic link existing between several **entities**. Finally, a **context network** is a graph of **contexts**. Within this graph, it is possible to go from a **context** to another on apparition/disappearance of **role** and/or **relation**. As said by Rey, this resulting model has been developed to be used in **HCI domain** with the goal to be a support tool in the development of new interactions methods.

In **HCI domain**, one key problem faced with **ubiquitous computing applications** is to have a good communication with the end-user. There are some situations where users can not be disturbed (even if they were, they would not be attentive [12]). Only information of a certain quality should be given to the user, otherwise, he would dismiss any kind of information. Some information might also be deferred for future notifications [11]. This implies that computers should try to understand users and their action in order to estimate which kind of information is relevant depending of the situation. The system needs the user to trust it; otherwise, it will not be able to react correctly. As an example, if a driver does not trust his car **Anti-lock Braking System** (**ABS**) and does not brake firmly, the braking efficiency will be decreased [4]. In order to establish a permanent communication between user and computer, interfaces are undoubtedly going to evolve.

To illustrate Rey's modelization, we will study a sample situation. The situation we are going to model is a classical presentation. In such a system, **relevant entities** are human beings and a pen (used as a pointer). In reality others entities exist: a video projector, a computer, a whiteboard but we will assume they are always present in the presentation room. This is not an unrealistic assumption and it will simplify the modelization process. As everyone

knows, a presentation needs the presence of at least two human beings[4]. Human can play the **roles** of speaker or of public. The pen can play the **role** of a pointer but a human can also points out something without using the pen, just with his hand or finger. This outlines each **role** can be played by one or several **entities**. It can also not been played by any **entity**. When held by a human, both **entities** (human and pen) are said in **relation**. As for **role**, **relations** may or may not be verified. A speaker can present something to an individual (2.4) or to a wider audience (2.5). During the presentation, anyone can point out important points with his finger (2.6) or with the pen (2.7).

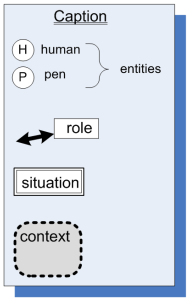

Figure 2.3: Common caption for Rey formalism

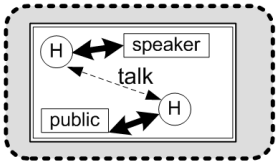

Figure 2.4: Presentation involving one speaker and one person in audience

[4]Here we assume training sessions are not relevant to our modelization

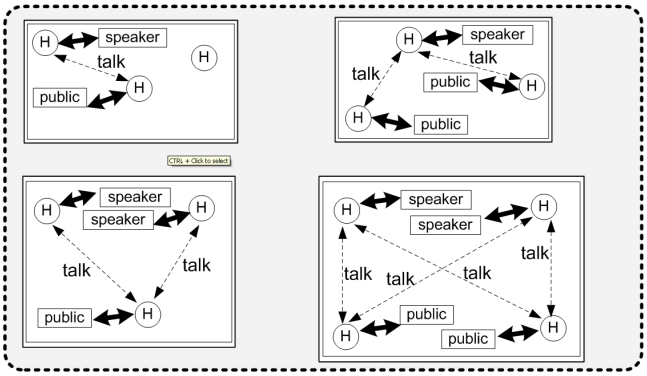

Figure 2.5: Presentation with multiple participants

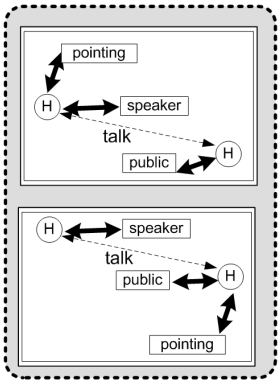

Figure 2.6: Pointing with hand

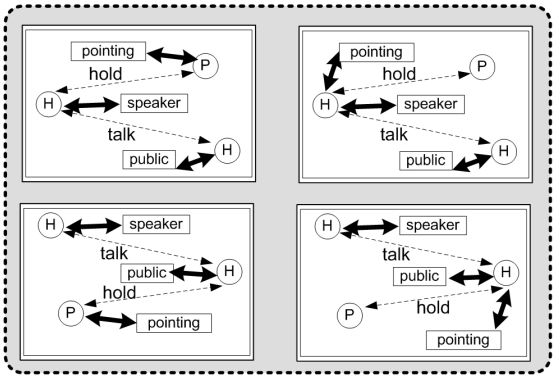

Figure 2.7: Pointing while holding a pen

2.2.2 The Physical Approach

The **physical approach** was designed to overcome the logical approach's drawbacks. Data are embedded within objects they apply to. Communications are following the **peer to peer** (**P2P**) communication model, which is more robust and naturally **fault-tolerant**. The main advantage of this approach is that it is totally independent of **device mobility**. If a device is moved, data automatically follows as they are embedded in the physical object.

According to Pauty, Couderc and Banâtre, the goal to achieve with a **physical approach** is to minimize the number of **explicit interactions** between man and machine [24]. As users are firstly human being, they already have activities outside the computing world. Computers can thus help human by automatically detecting the situations rather than asking users to explicitly enter

their **current situation** (**current context**) on an **input device**. By giving more responsibilities to the applications, users can concentrate on their **primary task**. To go a little bit further, if no **explicit interactions** are needed, device can interact more easily with ther devices as they will not be asked to take decisions.

As it does not rely on any existing **infrastructure**, it is best fitted for in the **field operation** [23]. However, in such a **decentralized system**, **universal knowledge** is quite impossible to obtain, so that decision are to be taken as locally as possible. Although the local decision may be taken more rapidly, it is also taken based on partial information and so can introduce inappropriate actions due to misinterpretations; but as everything is at a local scale, consequences are not as important as if a mistake was made in the logical approach as it may have an overall network impact.

The **physical approach** also has the benefit not to need any routing mechanisms as devices simply communicate with other "in range" devices. This can be illustrated with all system based on **tagged objects**. **Passive-tag** can be viewed as (very) simple device having a unique functionality that is to broadcast the unique **contextual data** they have, their **Universally Unique IDentifier** (**UUID**). All computation are done by the sensing device (**tag-reader**), which react to proximity or to apparition/disappearance of tags. Tags are hereby defining the **tag-reader** context.

To illustrate the **physical approach**, let us have a look to an aircraft safety mechanism developed to avoid in-flight collisions. The **Traffic alert and Collision Avoidance System** (**TCAS**) [19, 18] system is composed of a device placed in a plane. It has two main goals: detect possible collision and propose a solution to avoid it. Each plane periodically sends a vector modelling its position and its trajectory. Each **TCAS** system listens to incoming messages. If it computes an intersection between its couple position, trajectory and the data from the other aircraft, it triggers an alarm in both planes. More than simply informing them, both devices agree on a safe operation to do on both planes to avoid the collision. One plane is asked to increase its altitude whereas the other is asked to decrease its own altitude. Here context is used to increase safety.

The **physical approach** raises a challenging problem. If multiple applications need the same information from a unique **sensor**, direct communication between consumers and provider (no matter if data are polled or pushed) can be highly resource consuming. On the other hand, public broadcasting becomes a problem for **security** aspects. Thus, data multicasting protocols should to be both scalable and secured. This difficulty can be overcome more easilly when considering indirect context awareness rather than direct context awareness.

2.3 Direct versus Indirect Context Awareness

A **direct context-aware application** is considering context as an **input data**. It is handled as if it was a keyboard interactive input [22] or a file read from a

storage device [20].

Contrarily to a **direct context-aware application**, an **indirect one** uses context at a highest level. Context is used to help **service orchestration**. Depending on context, new services can be instantiated and others may be halted. Communications between all services of a **SOA** can be reorganized to adapt to the context's modification. The **indirect context-awareness** is taking place inside the middleware [8] in opposition to the **direct context-awareness** case for whom it take place at the application level. In this approach, **context** is compliant to the **loose coupling** principle of **SOA** as it is independent of **business code**. Context awareness is thus a general capability of the middleware orchestraction mechanism and not of each application.

2.4 Contextual Area

Most persons dealing with **context-aware computing** agree on a point, distance between two **entities** is important. In effect, a flying butterfly at the other side of the Earth is very unlikely to be taken into account for our local context. Those different granularities have already been outlined in [2]. In the **physical approach**, the **unified context** consist of mapping data directly into the physical space and to associate a **contextual area** in which this data will be accessible [24]. Actually **contextual areas**' shapes depend on the communication technology. It is mostly a sphere with radio transmission; but it can also be a cone with light communication (IR for example). Previous work on **sentient computing** with the **Bat system** [1] had a similar concept with the **Spatial Monitoring application**, but it was based on a **logical approach**.

In [16, 13, 14], Lavirotte, Lingrand and Tigli underline a key point in **cost functions** which is their asymmetric nature. As an illustration, it is easier to get down of a hill than to climb it up. More formally, it cost more (fuel, time, energy in general) to get up than to get down although the distance is the same. This suggests replacing previous symmetric concept of **distance functions** by the asymmetric concept of **cost functions**. On top of that, **cost functions** present another advantage: it becomes possible to define precise regions in space with any arbitrary shape. This is an significant improvement as previous works in this field were limited to regularly curved shapes (depending on technoloy aspects such as communication range). As **cost functions** can be arbitrarily defined, it leaves the possibility to define any arbitrary shape for **contextual area**.

Since **cost functions** are not any more symmetric, being in one's contextual area is not any more a symmetric relation. This means that if T is in C's **contextual area**, C may not be in T's **contextual area** ($T \in Z(C) C \in Z(T)$). This let us differentiate three kind of **selection mechanisms**. The **endo-selection** consists of selecting **entities** that a reference can see in its **local context** ($\{X/X \in Z(T)\}$). On the other hand, the **exo-selection** is the selection of all **entities** that see the reference in their own **distant context**. When both selections are verified at the same time ($T \in Z(C) \wedge C \in Z(T)$),

we qualify the relation as an **bilateral-selection**.

$$\text{endo}(R)=\{X/X \in Z(R)\}$$
$$\text{exo}(R)=\{X/R \in Z(X)\}$$
$$\text{bilateral}(R)=\{X/X \in \text{endo}(R) \cap \text{exo}(R)\} = \{X/X \in Z(R) \wedge X \in Z(R)\}$$

We will now recall an illustration (originally published in [14]) on the classical museum tour guide case illustrating this approach. We will consider three visitors A, B and C and a painting T. The **endo-selection** (Figure 2.8 a) consists of selecting all paintings visible by each visitor. Here we can see the problem of visitor C who sees a painting through a wall, such situation may effectively happen if we only look at the distance between two points without taking care of obstacles in their line of sight. The **exo-selection** (Figure 2.8 b) let us know all visitors that may see the painting. In fact, here again we have more than expected as B is looking in the opposite direction of painting T. The **exo-selection** may be useful in **contextual discovery** to only inform visitors of things they could effectively see. Finally, when we merge the results of **endo-selection** and **exo-selection**, we obtain the expected result with the **bilateral-selection** (Figure 2.8 c).

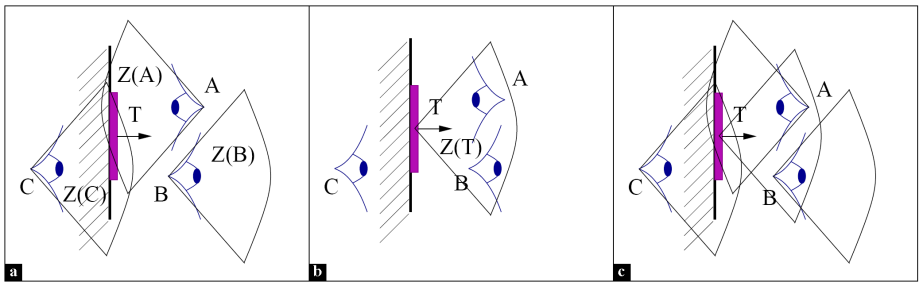

Figure 2.8:
a) **Endo-selection**: visitors A and C can see T
b) **Exo-selection**: T can be viewed by A and B
c) **Bilateral-selection**: Only A can see T

2.5 Towards Contextualized Services

Adding contextual capabilities to services need two different adaptations. Service discovery is the first mechanism to contextualize but context must also be checked afterwards to decide if communications should be upholded or not. As we have seen earlier, devices' communications are based use eventing mechanism. Eventing subscription need a similar filtering mechanism as the one used for discovery; events sending should also be contextualized as it is done for classical communications messages. Contextualization can be done in various ways: it can be done once or it may need to be coupled to a leasing mechanism. We will expose our proposed solution to add indirect context-aware capability in Web Service for Device frameworks

Chapter 3

Contribution

Although **services for devices** can be used to discover **devices** on a **local network** in most cases, it would be more **relevant** to discover **devices** in the **context** of the searcher, as defined in 2. This means that **context** must be known at some point, and we get it via **devices**.

We can thus distinguish two kinds of **devices**. Firstly, the classical devices are simple devices acting in the system as functional components. Secondly, some specific devices can provide **contextual data**. For example, a TV is a functional component which implements a display **device**, and a room-localization **device** would provide contextual data about it. This way, an application or an user will be able to locate TV which are in their **context**. To have more compact notations, we will shorten the non-contextual **devices** (it can be context-aware or not) by D and the **context provider** devices by C . D can rely on one or several C to obtain contextual information and each C may be used by multiples D . If a C is not used by any D we choose not to modelize them as if they are not used by any D , they do not play any **role** in the **application**.

One have to know how C are related to D . We put in place an **association table**, which associates each D **device** to a set of C devices. This table has to be hand-made. There is no way that **devices** can know how they will be used, in which **context**, with which **contextual informations** or from which **devices**. When creating an application from **WSD**, the table has to be hand-written as it is different for each **application**.

Several approaches are possible to deal with **context-aware devices**. We will study three of them, from the most simple but expensive, to a cleverer and more efficient one.

3.1 Simplest approach for context-aware devices

This approach is the simplest way to perform **contextual operations** towards **WSD**. In short, it checks context for every action, which is expensive in term

of messages and then **network bandwidth use**. Although not being very efficient, this solution guarantee that no operation will be done unless **context** is each time verified. It will be called case *A* in the comparison table in 3.5.

3.1.1 Contextual discovery

Context can be used at discovery time to only discover new **D** in the searcher's **contextual area**. The simplest way is to send a multicast search request for **D** , and for all **devices** responding, invocate a **context request** on all associated **C** using the association table. Then, a **device** can determine of which **devices** it is in the **context** of, and of which it is not (**exo selection**).

3.1.2 Contextual communication

Messages should only be exchanged between **D** if **context** is correct. Before invoking a method on a **D device**, associated **C devices** are queried for **context values**. If it appears the targetted **D device** is not anymore in the **context**, the method invocation is not made. Another similar **device** in the **context** should be searched.

3.1.3 Contextual events

keywordEvents can be partially **contextualized** by making a **contextual verification** (invoking a specific method to fetch the **context** on associated **C** devices) before the **subscription procedure** and afterwards by requiring a **leasing mechanism**. If the device gets out of the **context**, at the end of the lease, the **context** will be checked again for the re-subscription, and the subscription will be cancelled.

3.2 Extending existing WSD protocols to deal with context

Adding **context-awareness** to existing **Web Services** can only be done by adding a mechanism to exchange **contextual metadata** between **devices**. No matter the kind of **selection mechanism** (**endo**, **exo** or **bilateral**) nor the place where the **selection** is done (**consumer device**, **producer device**, **third part device**), exchanging specific data to handle **context** induce some additional cost to existing protocols.

3.2.1 Discovery

In proactive search (like **UPnP ssdp:discovery** and **WS-Discovery** Probe), **contextual meta-data** can be added to the broadcasted discovery message. Servers receiving this overloaded message can the proceed to **exo context selection** (from the searcher point of view), and reply to the search if the searcher

is in their **context**. This way, the **discovery** will only notify from **devices** for which **exo context** is valid, and reduce unneeded messages on the network.

Operation of this kind can also be done for **advertisement**, the (like **UPnP ssdp:alive** and **ssdp:byebye** or **WS-Discovery** Hello and Bye messages) when joining/leaving the network. Depending of the kind of **selection mechanism**, each device can send its **contextual data** or its **contextual evaluation function** ϕ. Receiving such overloaded messages, clients can proceed to an **endo**, **exo** or **bilateral selection** of the **context** for these **devices**, whithout make additional context requests.

3.2.2 Method invocation

We have seen that **WSD** still permit method invocations of original **Web Services**, using **SOAP**. However, if **devices** were discovered in the **context** and if they are moving or if the application is moving, they may not be in the **context** anymore when a method invocation is requested. What we do to ensure this, is adding **contextual informations** to method invocation requests. Receiving this, **contextual WSD** can choose not to process the request as usual if the **context** does not match, and send back an error instead of the return value.

3.2.3 Events

Two approaches can permit **contextualized eventing mechanisms**. The first simple solution is to check **context** during the **subscription mechanism**. In order to adapt to a potential evolution of **contextual data**, the **context** must be checked periodically. This can be done in combination with the subscription leases implemented in **WSD**.

On the other hand, each **event** can also be contextualized to improve **reactivity** and **security**. Nevertheless, it is not a simple thing to be done. Indeed, that imposes to know an up-to-date **context** of the subscribed clients. Such data can only be known by asking each **device** to provide it. This mechanism is a kind of **notification** of new **events**; each device still need to retrieve it and this can be filtered on context. However, such notifications may reveal a modification in the **device state** even if the remote device is not in the "**good context**". A quick fix to this unwanted **information disclosure** is to send some **fake notification**. A trade-off should be found between wasting **network bandwidth** and disclosing information to unauthorized devices. This trade-off is defined by the proportion of real/fake notification messages.

The principles presented here add **context capabilities** to generic **Web Service for Device** but some improvements can reduce the overhead cost induced by all the **contextual metadata messages** exchanges. Some architectural modification are thus presented in the next section.

3.3 Architectural improvements for Contextual WSOAD efficiency

When adding **context** to existing **Web Services for Devices**, two strategies can be adopted. Firstly we can privilege **interoperability** with existing non **context-aware** servers and admit that they can be used in the new system. On the other hand, if we want to consider **security** as part of the **context**, out-of-context servers must not be discovered and interactions must be restricted to situations in which the **security context** is verified.

3.3.1 Devices aggregation

To minimize **network communication costs**, some **devices** can be grouped together. Each of our groups contain one or several **D** and their associated **C** . If a **C** is used by several **D** , all will be in the same **aggregate** as illustrated in figure 3.1. Statistically, if each **aggregate** contains several **C** , **context** can be asked only once to the **aggregate** and not individually to each **C** . This kind of batch processing helps in decreasing communications costs induced by the **contextual queries**. **Aggregates** will be noted as case *B* in the comparison table in 3.5.

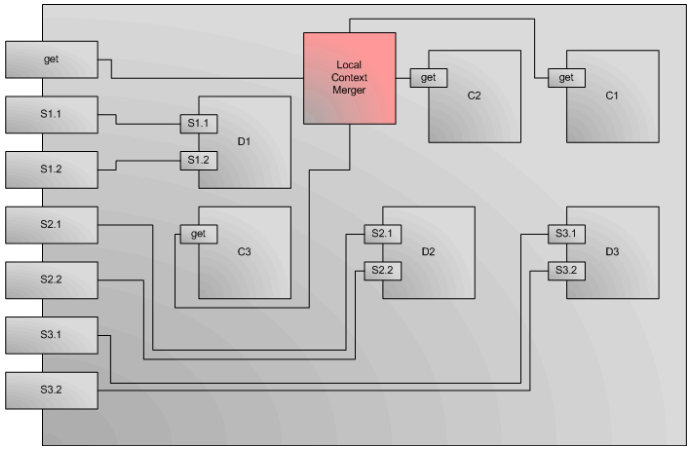

Figure 3.1: Devices aggregation

3.3.2 Aggregating devices using type

keywordDevice Aggregation!Typed Device Aggregation **WSD** protocols provide a **typed search mechanism** which can be used to build new **aggregates**. As this is only provided for search, it is not helpful for **communication** nor **eventing**. This is illustrated in figure 3.2

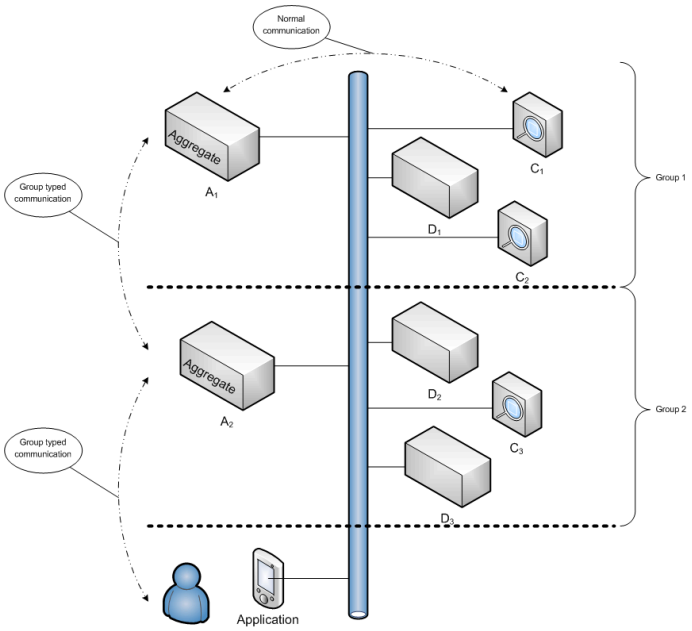

Figure 3.2: Typed solution

3.3.3 Aggregating devices in virtual networks

Another solution can be built on top of **virtual networks**; each aggregate is assigned an unique network address. Therefore, each **device** have two addresses: the **physical one** inherited from its host and the **virtual one** defining in which aggregate it belongs.

3.3.3.1 Aggregating devices on the localhost address

We can consider each aggregate as a **local network**. Each device is thus started on the **localhost network (127.0.0.1)** as shown on figure 3.3. This impose a strict limit of at most one aggregate on each physical host, and the ability to specify the network interface to be bound to, for services as well as clients.

3.3.3.2 Aggregating devices on a virtual IP address

A simple modification to the previous solution permits multiple **aggregates** on the same **host**. Instead of a **localhost IP address**, we can give a specific **virtual IP address**. This also gives the ability to split an **aggregate** on several physical hosts. From a **security** point of view, this can be viewed as an open system as the original services can still be accessed on their **virtual IP address** without any **contextual verifications**. An important thing to consider in this solution is the procedure to assign each virtual address to each

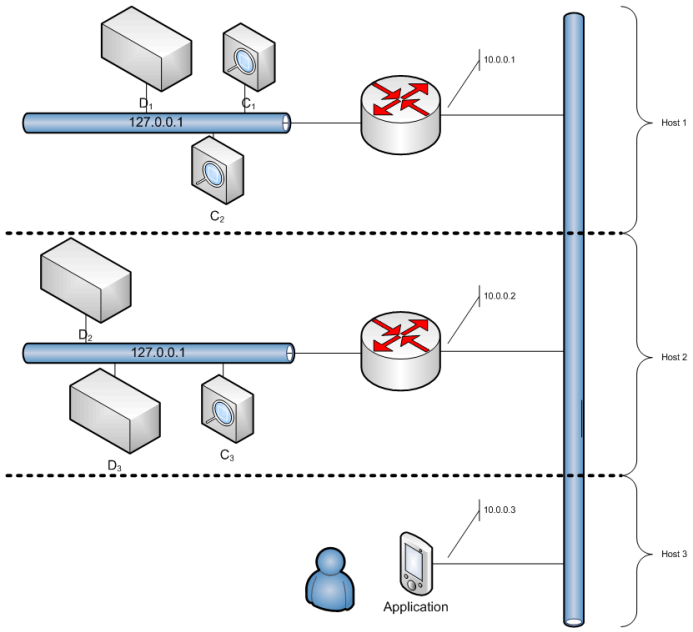

Figure 3.3: IP solution

device. This configuration of each **Web Service** IP is not a trivial problem as it has to be universally unique but also known by every implied service in the interactions.

3.3.4 Context-Aware Bridge Solution

This solution is apparented to a network **DeMilitarized Zone** (**DMZ**). We authorize service to operate as usual inside the **DMZ** but we build a strong barrier around them to protect them from the rest of the world. All communication with the outside are thus handled in an unique point: the **bridge**. This **bridge** is responsible not to forward messages if the **contextual metadata** are absent or invalid.

The desired architecture is shown in figure 3.4 on the left-hand side. Reality (right-hand side) is a little bit different. The difference has to be underlined as it reveal a potential security problem. All **devices** on the right are all member of the same **physical network** (illustrated with the widest cloud); nothing can prevent a **malicious device** from listening the **network** and catching data packets thus bypassing the **contextual filtering mechanism**.

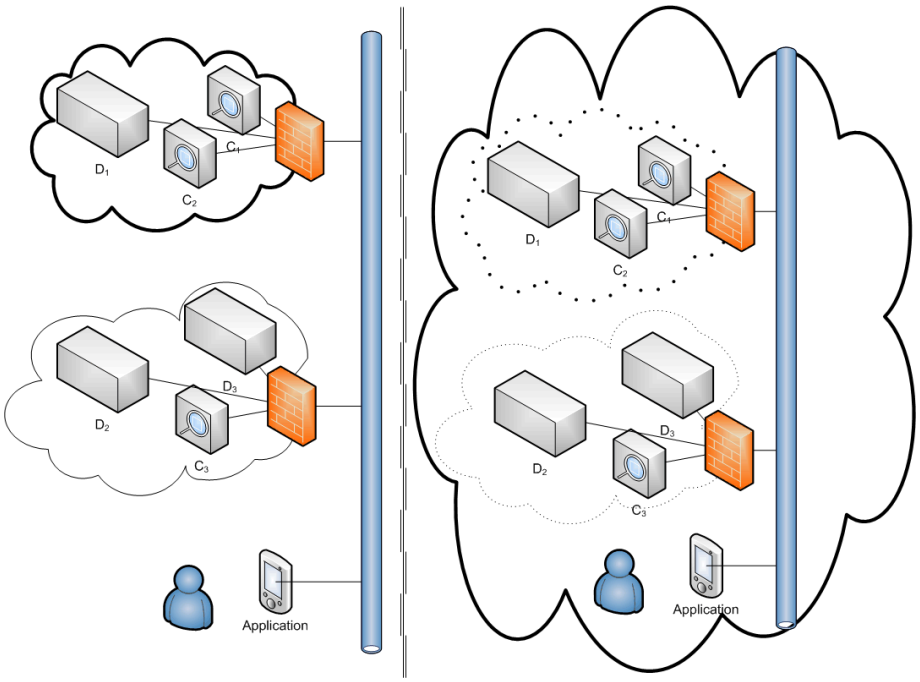

Figure 3.4: Bridge solution: at left, what we want; at right, what reality looks like

3.4 Efficient approach: contextual WSD and device aggregates

The best approach we can present for **contextual WSD** handling is using both features presented in 3.2 and 3.3, protocol modification and **device aggregation**. Device aggregation helps in reducing **contextual metadata** overhead costs. Protocols need to be adapted firstly to handle context but it can also be usefull to increase **security**, as **non-context-aware protocols** were not designed to protect contextual privacy for example. This will be the **C** case of our comparison table in next section.

3.5 Cost Evaluation

To summarize somehow all **contextual WSD** approaches we have explained, we will compare costs in packets number for each operation (**discovery**, **method invocation** and **eventing**) in table 3.1. We first need to define variables we will use.

Case	Discovery		Invocation on device i		Events on device i	
	Research	Context request	Context	Invocation	Subscription	Event
A	$\gamma + \alpha N$	$(\beta_1 + \beta_2)K$	$(\beta_1 + \beta_2)\tau_i$	$\beta_1 + \beta_2$	$(\beta_1 + \beta_2)\tau_i + \beta_3 + \beta_4$	$\beta_5 + \beta_6$
B	$\gamma + \alpha N$	$(\beta_1 + \beta_2)N$	$\beta_1 + \beta_2$	$\beta_1 + \beta_2$	$\beta_1 + \beta_2 + \beta_3 + \beta_4$	$\beta_5 + \beta_6$
C	$\gamma' + \alpha r N$	0	0	$\beta'_1 + \beta_2$	$\beta'_3 + \beta_4$	$\beta_5 + \beta_6$

Table 3.1: Costs of Communication function of Case

We take a set of N devices. In that set we count P **D** devices, and K **C** devices. We can thus write $N = P + K$. A **D** device has probably more than one **C** device to refer to to get its context, and we note this number τ_i.

When dealing with **aggregated devices**, N is the number of aggregates. They are composed of p_i **D** and optionnaly k_i **C** devices.

In our modelisation, the base unity will be the number of network packets needed to achieve **contextual operations** on **WSD**. Lots of variables depend on the **WSD** infrastructure protocol: α, γ, γ' and all β_k and β'_k. γ is the number of packets used to send a multicast research request, and γ' is the same thing but in the case of a contextual research request: they contain **context information**. α is the number of packets a **WSD** uses to respond to such a request. β_1 is the number of packets needed to make a remote method invocation, β'_1 is the same with context information, β_2 is the number of packets for the response. These numbers are counting communication establishment packets, acknowledgements at all levels, and connection terminations. β_3 is the number of packets needed for a **subscription request**, β'_3 for a **contextual subscription request**, and β_4 is the number of packets of the subscription acceptance or rejection. β_5 is the number of packets used to send an event, and β_6 the number of packets used for the acknowledgement of the event.

We note r the ratio representing the number of **devices** belonging to the **context** of the application ($0 \leq r \leq 1$).

We set the hypothesis that **C** transmit the complete contextual information in a unique request packet.

The case A is the one presented in 3.1, the obvious and inefficient one. The case B is a solution with aggregates, presented in 3.3. Finally, the case C is the best solution, prensented in 3.4.

Table 3.1 represents communication costs on successfull **context matching** in three different **context handling cases**. If not matching, invocation's cost is null for cases A, B, and subscription is not done so $\beta_3 + \beta_4 = 0$ and there is no event sent.

Chapter 4

Conclusion

We have seen that protocols needed to be adapted to handle context. On the basis of the current standards of Web services, we need to modify existing protocols or specify a new added protocol to deal with context. Moreover such context aware mechanisms for web services must be implemented in the both discovery and communication phases on the life cyle of the corresponding web service. In case of eventing communications, we can notice an interesting similarity with research and discovery phase.

Indeed subscribing to a devices' event channel is similar to a publication: the event consumer publishes its interest in such events. Afterwards, each event sent should be contextualized as normal communications, the difference being that sender's and receiver's role are exchanged. An interesant thing to notice from this symmetry is that both selection mode can be used. An endo-selection filtering mechanism at subscription time implies to have an exo-selection filtering mechanism when sending an event. The same remains true for the classical discovery and communications phases. Implementations can send periodically values of local context (leasing mechanism) to transform an endo (resp. exo)-selection mechanism into an exo (resp. endo)-selection mechanism, as explained in this report.

In conclusion, this report is a first step in the investigation to well-defined and validate a new protocol for context awarness of web services and the corresponding applications in the field of the context dependent access to device for security in collaboration with the spohipolitan MobileGov company. First results will be communicated in CMMSE Workshop in Lyon at the end of this year.

Concepts

Keywords

Bibliography

[1] Mike Addlesee, Rupert Curwen, Steve Hodges, Joe Newman, Pete Steggles, Andy Ward, and Andy Hopper. Implementing a sentient computing system. 08 2001.

[2] Patrick Brézillon. Hors du contexte, point de salut. page 5, 02 2002.

[3] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. page 16, 11 2000.

[4] Keith Cheverst, Nigel Davies, Keith Mitchell, and Christos Efstratiou. Using context as a crystal ball: Rewards and pitfalls. page 5, 2000.

[5] Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. *Context is key*, volume 48 of *3*. 03 2005.

[6] Joëlle Coutaz and Gaëtan Rey. Foundations for a theory of contextors. page 22, 2002.

[7] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, 11 2000.

[8] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. page 67, 2001.

[9] Jérôme Godard. *Modeling and Services for Adaptive Collaborative Delivery of Annotated Multimedia Resources*. PhD thesis, The Graduate University for Advanced Studies, 03 2005.

[10] Edward Twitchell Hall. Beyond culture. 1976.

[11] Eric Horvitz, Andy Jacobs, and David Hovel. Attention-sensitive alerting. page 10, 1999.

[12] Scott E. Hudson, James Fogarty, Christopher G. Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny C. Lee, and Jie Yang. Predicting human interruptibility with sensors: A wizard of oz feasibility study. page 8, 2003.

[13] Stéphane Lavirotte, Diane Lingrand, and Jean-Yves Tigli. Définition du contexte: Fonctions de coût et méthodes de sélection. In *Proceedings of the Deuxième Journée Francophone: Mobilité et Ubiquité 2005*, pages 9–12, Grenoble, France, 06 2005.

[14] Stéphane Lavirotte, Diane Lingrand, and Jean-Yves Tigli. A propos du contexte et des différentes méthodes de sélection associées. Technical report, I3S - Informatique, Signaux et Systèmes de Sophia Antipolis - UMR 6070, 02 2005.

[15] Stéphane Lavirotte and Loïc Pottier. Optical formula recognition. 1997.

[16] Diane Lingrand, Stéphane Lavirotte, and Jean-Yves Tigli. Selection using non symmetric context areas. LNCS 3762:225–228, 10 2005.

[17] Ramiro Liscano. Service discovery in sensor networks: An overview. page 51, 2003.

[18] Carolos Livadas, John Lygeros, and Nancy A. Lynch. High-level modeling and analysis of tcas. page 11, 1999.

[19] John Lygeros and Nancy Lynch. On the formal verification of the tcas conflict resolution algorithms. page 6, 1997.

[20] Ian MacColl, Barry Brown, Steve Benford, Matthew Chalmers, Ruth Conroy, Nick Dalton, Areti Galani, Chris Greenhalgh, Danius Michaelides, Dave Millard, Cliff Randell, Tom Rodden, Anthony Steed, Ian Taylor, and Mark Weal. Shared visiting in equator city. 2002.

[21] Yoosoo Oh, Choonsung Shin, Seiie Jang, and Woontack Woo. ubi-ucam 2.0: A unified context-aware application model for ubiquitous computing environments. 06 2005.

[22] Jason Pascoe. The stick-e note architecture: Extending the interface beyond the user. page 4, 1997.

[23] Jason Pascoe, Nick Ryan, and David Morse. Human-computer-giraffe interaction: Hci in the field. page 10, 1998.

[24] Julien Pauty, Paul Couderc, and Michel Banâtre. Synthèse des méthodes de programmation en informatique contextuelle. Technical report, 01 2004.

[25] Gaëtan Rey. Contexte en interaction homme-machine : le contexteur. page 127, 2005.

[26] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. page 6, 1994.

[27] Ali ShaikhAli, Omer F. Rana, Rashid Al-Ali, and David W. Walker. Uddie: An extended registry for web services. pages 85–89, 01 2003.

[28] M. Weiser, R. Gold, and J. B. Brown. The origins of ubiquitous computing research at parc in the late 1980s. 38(4), 1999.