

## Interoperability in Complex Distributed Systems

Gordon S. Blair<sup>1</sup>, Massimo Paolucci<sup>2</sup>, Paul Grace<sup>1</sup>, Nikolaos Georgantas<sup>3</sup>

<sup>1</sup>School of Computing and Communications, Lancaster University, UK  
{gordon, gracep}@comp.lancs.ac.uk

<sup>2</sup>Laboratories Europe GmbH, Munich, Germany  
paolucci@docomolab-euro.com

<sup>3</sup>INRIA, CRI Paris-Rocquencourt, France  
nikolaos.georgantas@inria.fr

**Abstract.** Distributed systems are becoming more complex in terms of both the level of heterogeneity encountered coupled with a high level of dynamism of such systems. Taken together, this makes it very difficult to achieve the crucial property of interoperability that is enabling two arbitrary systems to work together relying only on their declared service specification. This chapter examines this issue of interoperability in considerable detail, looking initially at the problem space, and in particular the key barriers to interoperability, and then moving on to the solution space, focusing on research in the middleware and semantic interoperability communities. We argue that existing approaches are simply unable to meet the demands of the complex distributed systems of today and that the lack of integration between the work on middleware and semantic interoperability is a clear impediment to progress in this area. We outline a roadmap towards meeting the challenges of interoperability including the need for integration across these two communities, resulting in middleware solutions that are intrinsically based on semantic meaning. We also advocate a dynamic approach to interoperability based on the concept of emergent middleware.

**Keywords:** Interoperability, complex distributed systems, heterogeneity, adaptive distributed systems, middleware, semantic interoperability

### 1 Introduction

Complex pervasive systems are replacing the traditional view of homogenous distributed systems, where domain-specific applications are individually designed and developed upon domain-specific platforms and middleware, for example, Grid applications, Mobile Ad-hoc Network applications, enterprise systems and sensor networks. Instead, these technology-dependent islands are themselves dynamically composed and connected together to create richer interconnected structures, often referred to as systems of systems. While there are many challenges to engineering such complex distributed systems, a central one is ‘interoperability’, i.e., the ability for one or more systems to: connect, understand and exchange data with one another for a given purpose. When considering interoperability there are two key properties to deal with:

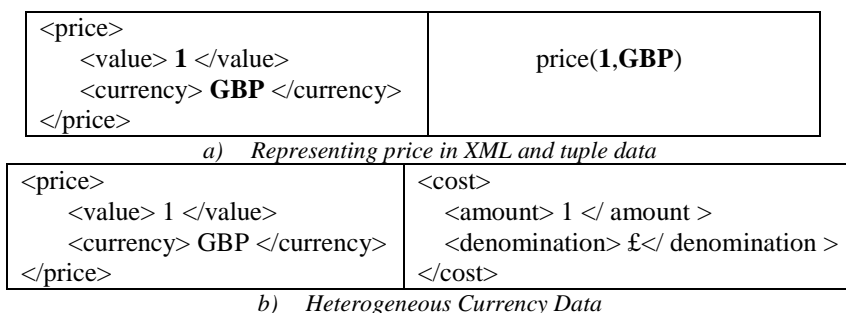
- *Extreme heterogeneity.* Pervasive sensors, embedded devices, PCs, mobile phones, and supercomputers are connected using a range of networking solutions, network protocols, middleware protocols, and application protocols and data types. Each of these can be seen to add to the plethora of technology islands, i.e., systems that cannot interoperate.
- *Dynamic and spontaneous communication.* Connections between systems are not made until runtime; no design or deployment decision, e.g., the choice of middleware, can inform the interoperability solution.

We highlight in this chapter the important dimensions that act as a barrier to interoperability; these consist of differences in: the data formats and content, the application protocols, the middleware protocols and the non-functional properties. We then investigate state-of-the-art solutions to interoperability from the middleware and the semantic web community. This highlights that the approaches so far are not fit for purpose, and importantly that the two communities are disjoint from one another. Hence, we advocate that the two fields embrace each other’s results, and that from this, fundamentally different solutions will emerge in order to drop the interoperability barrier.

## 2 Interoperability Barriers: Dimensions of Heterogeneity

### 2.1 Data Heterogeneity

Different systems choose to represent data in different ways, and such data representation heterogeneity is typically manifested at two levels. The simplest form of data interoperability is at the syntactic level where two different systems may use two very different formats to express the same information. Consider a vendor application for the sale of goods; one vendor may price an item using XML, while another may serialize its data using a Java-like syntax. So the simple information that the item costs £1 may result in the two different representations as shown in Fig. 1(a).



**Fig. 1.** Examples of Data Heterogeneity

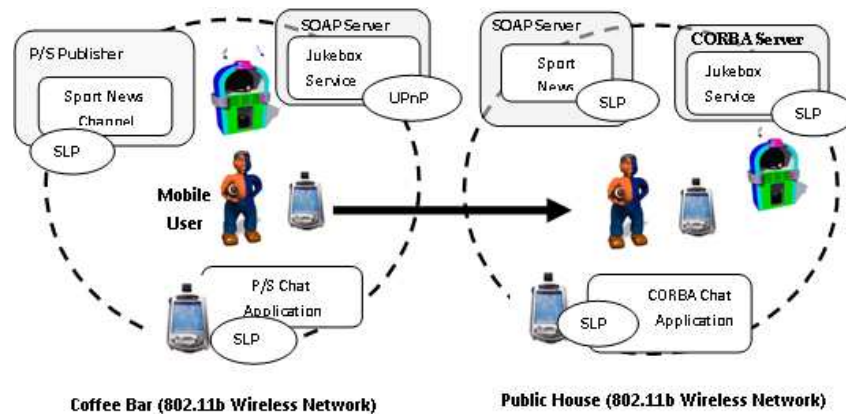
Aside from the syntactic level interoperability, there is a greater problem with the “*meaning*” of the tokens in the messages. Even if the two components use the same

syntax, say XML, there is no guarantee that the two systems recognize all the nodes in the parsing trees or even that the two systems interpret all these nodes in a consistent way. Consider the two XML structures in the example in Fig. 1(b). Both structures are in XML and they (intuitively) carry the same meaning. Any system that recognizes the first structure will also be able to parse the second one, but will fail to recognize the similarity between them unless the system realizes that *price=cost*, that *value=amount*, that *currency=denomination* and of course that *GBP=£* (where  $\equiv$  means equivalent). The net result of using XML is that both systems will be in the awkward situation of parsing each other's message, but not knowing what to do with the information that they just received.

The deeper problem of data heterogeneity is the semantic interoperability problem whereby all systems provide the same interpretation to data. The examples provided above, show one aspect of data interoperability, namely the recognition that two different labels represent the same object. This is in the general case an extremely difficult problem which is under active research [1], though in many cases it can receive a simple pragmatic solution by forcing the existence of a shared dictionary. But the semantic interoperability problem goes beyond the recognition that two labels refer to the same entity. Ultimately, the data interoperation problem is to guarantee that all components of the system share the same understanding of the data transmitted, where the same understanding means that they have consistent semantic representations of the data.

## 2.2 Middleware Heterogeneity

Developers can choose to implement their distributed applications and services upon a wide range of middleware solutions that are now currently available. In particular, these consist of heterogeneous *discovery* protocols which are used to locate or advertise the services in use, and *heterogeneous interaction* protocols which perform the direct communication between the services. Fig. 2 illustrates a collection implemented upon these different technologies. Application 1 is a mobile sport news application, whereby news stories of interest are presented to the user based on their current location. Application 2 is a jukebox application that allows users to select and play music on an audio output device at that location. Finally, application 3 is a chat application that allows two mobile users to interact with one another. In two locations (a coffee bar and a public house) the same application services are available to the user, but their middleware implementations differ. For example, the Sport News service is implemented as a publish-subscribe channel at the coffee bar and as a SOAP service in the public house. Similarly, the chat applications and jukebox services are implemented using different middleware types. The service discovery protocols are also heterogeneous, i.e., the services available at the public house are discoverable using SLP and the services at the coffee bar can be found using both UPnP and SLP. For example, at the coffee bar the jukebox application must first find its corresponding service using UPnP and then use SOAP to control functionality. When it moves to the public house, SLP and CORBA must be used.



**Fig. 2.** Legacy services implemented using heterogeneous middleware

### 2.3 Application Heterogeneity

Interoperability challenges at the application level might arise due to the different ways the application developers might choose to implement the program functionality, including different use of the underlying middleware. As a specific example, a merchant application could be implemented using one of two approaches for the consumer to obtain information about his wares:

- A single `GetInfo()` remote method, which returns the information about the product price and quantity available needed by the consumer.
- Two separate remote methods `GetPrice()`, and `GetQuantity()` returning the same information.

A client developer can then code the consumer using either one of the approaches described above, and this would lead to different sequences of messages between the consumer and merchant. Additionally, application level heterogeneity can also be caused due to the differences between the underlying middlewares. For example, when using a Tuple Space, the programmer can use the rich search semantics provided by it, which are not available in other types of middleware, e.g., for RPC middleware a Naming Service or discovery protocol must then be used for equivalent capabilities.

### 2.4 Non-Functional Heterogeneity

Distributed systems have non-functional properties that must also be considered if interoperability is to be achieved. That is, two systems may be able to overcome all of the three prior barriers and functionally interoperate, but if the solution does not satisfy the non-functional requirements of each of the endpoints then it cannot be considered to have achieved full interoperability. For example, peers may have different requirements for the latency of message delivery; if the client requires that

messages be delivered within 5ms and the server can only achieve delivery in 10ms then interoperability is not satisfying the solution. Similarly, two systems may employ different security protocols; the interoperability solution must ensure that the security requirements of both systems are maintained.

### 3 Middleware Solutions to Interoperability

#### 3.1 Introduction

Tanenbaum and Van Steen define interoperability as:

*“the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard.”* [2].

Achieving such interoperability between independently developed systems has been one of the fundamental goals of middleware researchers and developers. This section traces these efforts looking at traditional middleware that seek a common standard/platform for the entire distributed system (section 3.2), interoperability platforms that recognize that middleware heterogeneity is inevitable and hence allows clients to communicate with a given middleware as dynamically encountered (section 3.3), software bridges that support the two-way translation between different middleware platforms (section 3.4), transparent interoperability solutions that go beyond interoperability platforms by allowing two legacy applications to transparently communicate without any change to these applications (section 3.5), and finally the logical mobility approach that overcomes heterogeneity by migrating applications and services to the local environment, assuming that environment has the mechanisms to interpret this code, e.g. through an appropriate virtual machine.

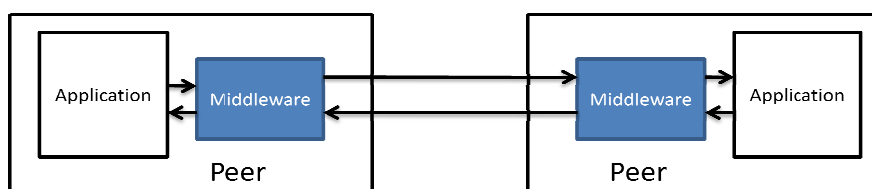
#### 3.2 Traditional Middleware

The traditional approach to resolving interoperability using middleware standards and platforms is to advocate that all systems are implemented upon the same middleware technology; this pattern is illustrated in Fig. 3 and is equivalent to native spoken language interoperability where the speakers agree in advance upon one language to speak. There are many different middleware styles that follow this pattern of interoperability, and it is important to highlight that these actually contribute to the interoperability problem, i.e., the different styles and specific implementations do not interoperate; to illustrate this point the following is a list of the most commonly used solution types:

- *RPC/Distributed Objects*. Distributed Objects (e.g. CORBA [3] and DCOM [4]) are a communication abstraction where a distributed application is decomposed into objects that are remotely deployed and communicate and co-ordinate with one another. The abstraction is closely related to the well-established methodology of object orientation, but rather than method invocations between

local objects, distributed objects communicate using remote method invocations; where a method call and parameters are marshalled and sent across the network and the result of the method is returned synchronously to the caller. This is similar to the style of communication employed in remote procedure calls (RPC) e.g. SunRPC [5].

- *Message-based*. Messaging applications differ from RPC in that they provide a one-way, asynchronous exchange of data. This can either be i) direct between two endpoints e.g. SOAP messaging, or ii) involve an intermediary element such as a message queue that allows the sender and receiver to be decoupled in time and space i.e. both do not need to be connected directly or at the same time. Examples of message queue middleware are MSMQ [6] and the Java Messaging Service (JMS)<sup>1</sup>.
- *Publish-Subscribe* is an alternative messaging abstraction where the producers and consumers of messages are anonymous from one another. Consumers subscribe for content by publishing a subscription (this can be topic-based, i.e., based upon the type of the message, or content-based i.e. the filter is fine-grained towards the content of each message); and publishers then send out messages/events. Brokers are intermediary systems that are deployed in the network or at the edge which match messages to subscriptions. A match then requires the event to be delivered to the application. Notable examples of Publish-Subscribe middleware are SIENA [7] and JMS.
- *Tuple Spaces*. The Linda platform [8] originated the concept of tuple spaces as a shared-memory approach for the coordination of systems. Clients can write and read data tuples into a shared space, where a tuple is a data element much like a database record. Tuple space middleware often differ in how the tuple space is deployed e.g. enterprise solutions, such as TSpaces [9], use a central enterprise server to host the tuple space for clients to connect to, while L<sup>2</sup>imbo [10] and LiME[11] distribute the tuple space evenly among peers.



**Fig. 3.** Interoperability pattern utilised by traditional middleware

These technologies resolve interoperability challenges to different extents, the majority focusing on interoperation between systems and machines with heterogeneous hardware and operating systems, and applications written in different programming languages. Hence, this pattern works well for distributed systems where the parties and technologies are known in advance and can be implemented using a common middleware choice. However, for pervasive and dynamic environments where systems interact spontaneously this approach is infeasible (every application

<sup>1</sup> <http://www.oracle.com/technetwork/java/jms/index.html>

would be required to be implemented upon the same middleware). In the more general sense of achieving universal interoperability and dynamic interoperability between spontaneous communicating systems they have failed. Within the field of distributed software systems, any approach that assumes a common middleware or standard is destined to fail due to the following reasons:

1. A one size fits all standard/middleware cannot cope with the extreme heterogeneity of distributed systems, e.g. from small scale sensor applications through to large scale Internet applications. CORBA and Web Services [12] both present a common communication abstraction i.e. distributed objects or service orientation. However, the list of diverse middleware types already illustrates the need for heterogeneous abstractions.
2. New distributed systems and application emerge fast, while standards development is a slow, incremental process. Hence, it is likely that new technologies will appear that will make a pre-existing interoperability standard obsolete, c.f. CORBA versus Web Services (neither can talk to the other).
3. Legacy platforms remain useful. Indeed, CORBA applications remain widely in use today. However, new standards do not typically embrace this legacy issue; this in turn leads to immediate interoperability problems.

### 3.3 Interoperability Platforms

Fig. 4 illustrates the pattern employed by *interoperability platforms*, which can be seen to follow the spoken language translation approach of the person speaking another person's language. Interoperability platforms provide a middleware-agnostic technology for client, server, or peer applications to be implemented directly upon in order to guarantee that the application can interoperate with all services irrespective of the middleware technologies they employ. First, the interoperability platform presents an API for developing applications with. Secondly, it provides a substitution mechanism where the implementation of the protocol to be translated to, is deployed locally by the middleware to allow communication directly with the legacy peers (which are simply legacy applications and their middleware). Thirdly, the API calls are translated to the substituted middleware protocol. A key feature of this approach is that it does not require reliance on interoperability software located elsewhere, e.g., a remote bridge, an infra-structure server, or the corresponding endpoint; this makes it ideal for infra-structureless environments. For the particular use case, where you want a client application to interoperate with everyone else, interoperability platforms are a powerful approach. These solutions rely upon a design time choice to develop applications upon the interoperability platforms; therefore, they are unsuited to other interoperability cases, e.g., when two applications developed upon different legacy middleware want to interoperate spontaneously at runtime. We now discuss three key examples of interoperability platforms.

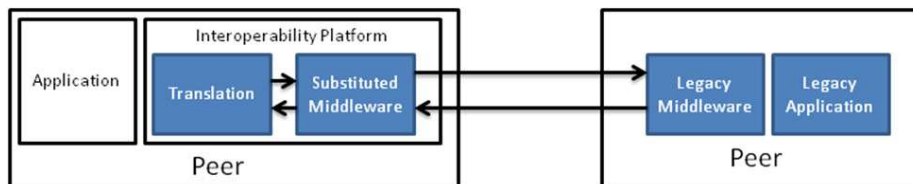


Fig. 4. Interoperability pattern utilised by interoperability platforms

**Universally Interoperable Core (UIC)** [13] was an early solution to the middleware interoperability problem; in particular it was an adaptive middleware whose goal was to support interactions from a mobile client system to one or more types of distributed object solutions at the server side, e.g., CORBA, SOAP and Java RMI. The UIC implementation was based upon the architectural strategy pattern of the dynamicTAO system [14]; namely, a skeleton of abstract components that form the base architecture is then specialised to the specific properties of particular middleware platforms by adding middleware specific components to it (e.g. a CORBA message marshaller and demarshaller).

**ReMMoC** [15] is an adaptive middleware developed to ensure interoperability between mobile device applications and the available services in their local environment. Here, two phases of interoperability are important: i) discovery of available services in the environment, and ii) interaction with a chosen service. The solution is a middleware architecture that is employed on the client device for applications to be developed upon. It consists of two core frameworks. A service discovery framework is configured to use different service discovery protocols in order to discover services advertised by those protocols; a complete implementation of each protocol is plugged into the framework. Similarly, a binding framework allows the interaction between services by plugging-in different binding type implementations, e.g., an IIOP client, a publisher, a SOAP client, etc.

The **Web Services Invocation Framework (WSIF)** [16] is a Java API, originating at IBM and now an Apache release, for invoking Web Services irrespective of how and where these services are provided. Its fundamental goal is to achieve a solution to better client and Web Service interoperability by freeing the Web Services Architecture from the restrictions of the SOAP messaging format. WSIF utilises the benefits of discovery and description of services in WSDL, but applied to a wider domain of middleware, not just SOAP and XML messages. The structure of WSDL allows the same abstract interface to be implemented by multiple message binding formats, e.g., IIOP and SOAP; to support this, the WSDL schema is extended to understand each format. The core of the framework is a pluggable architecture into which providers can be placed. A provider is a piece of code that supports each specific binding extension to the WSDL description, i.e., the provider uses the specification to map an invoked abstract operation to the correct message format for the underlying middleware.

### 3.4 Software Bridges

Software bridges enable communication between different middleware environments. Hence, clients in one middleware domain can interoperate with servers in another middleware domain. The bridge acts as a one-to-one mapping between domains; it will take messages from a client in one format and then marshal this to the format of the server middleware; the response is then mapped to the original message format. Fig. 5 illustrates this pattern, which can be seen as equivalent to employing a translator to communicate between native speakers. Many bridging solutions have been produced between established commercial platforms. The OMG has created the DCOM/CORBA Inter-working specification [17] that defines the bi-directional mapping between DCOM and CORBA and the locations of the bridge in the process. SOAP2CORBA<sup>2</sup> is an open source implementation of a fully functional bi-directional SOAP to CORBA Bridge. While a recognised solution to interoperability, bridging is infeasible in the long term as the number of middleware systems grow, i.e., due to the effort required to build direct bridges between all of the different middleware protocols.

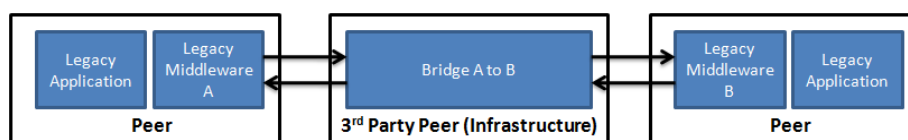


Fig. 5. Interoperability pattern utilised by Software Bridges

**Enterprise Service Buses (ESB)** can be seen as a special type of software bridge; they specify a service-oriented middleware with a message-oriented abstraction layer atop different messaging protocols (e.g., SOAP, JMS, SMTP). Rather than provide a direct one-to-one mapping between two messaging protocols, a service bus offers an intermediary message bus. Each service (e.g. a legacy database, JMS queue, Web Service etc.) maps its own message onto the bus using a piece of code, to connect and map, deployed on the peer device. The bus then transmits the intermediary messages to the corresponding endpoints that reverse the translation from the intermediary to the local message type. Hence traditional bridges offer a 1-1 mapping; ESBs offer an N-1-M mapping. Example ESBs are Artix<sup>3</sup> and IBM Websphere Message Broker<sup>4</sup>.

Bridging solutions have shown techniques whereby two protocols can be mapped onto one another. These can either use a one-to-one mapping or an intermediary bridge; the latter allowing a range of protocols to easily bridge between one another. This is one of the fundamental techniques to achieve interoperability. Furthermore, the bridge is usually a known element that each of the end systems must be aware of and connect to in advance-again this limits the potential for two legacy-based applications to interoperate.

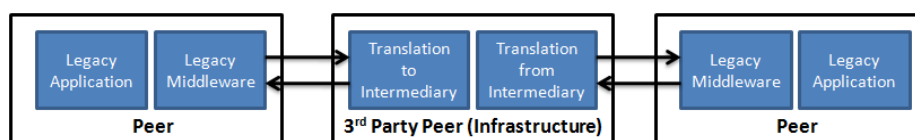
<sup>2</sup> <http://soap2corba.sourceforge.net>

<sup>3</sup> <http://web.progress.com/en/sonic/artix-index.html>

<sup>4</sup> <http://www-01.ibm.com/software/integration/wbimessagebroker/>

### 3.6 Transparent Interoperability

In transparent interoperability neither legacy implementation is aware of the encountered heterogeneity, and hence legacy applications can be made to communicate with one another. Fig. 6 shows the key elements of the approach. Here, the protocol specific messages, behaviour and data are captured by the interoperability framework and then translated to an intermediary representation (note the special case of a one-to-one mapping, or bridge is where the intermediary is the corresponding protocol); a subsequent mapper then translates from the intermediary to the specific legacy middleware to interoperate with. The use of an intermediary means that one middleware can be mapped to any other by developing these two elements only (i.e. a direct mapping to every other protocol is not required). Another difference to bridging is that the peers are unaware of the translators (and no software is required to connect to them, as opposed to connecting applications to 'bridges'). There are a number of variations of this approach, in particular where the two parts of the translation process are deployed. They could be deployed separately or together on one or more of the peers (but in separate processes transparent to the application); however, they are commonly deployed across one or more infrastructure servers.



**Fig. 6.** Interoperability pattern utilised by Transparent Interoperability Solutions

There are four important examples of transparent interoperability solutions:

- *The INteroperable Discovery System for networked Services (INDISS)* system [18] is a service discovery middleware based on event-based parsing techniques to provide service discovery interoperability in home networked environments. INDISS subscribes to several SDP multicast groups and listens to their respective ports. To then process the incoming raw data flow INDISS uses protocol specific parsers, which are responsible for translating the data into a specific message syntax (e.g. SLP) and then extracting semantic concepts (e.g. a lookup request) into an intermediary event format. Events are then delivered to composers that translate this event to the protocol specific message of (e.g. UPnP) the protocol to interoperate with.
- *uMiddle* [19] is a distributed middleware infrastructure that ties devices from different discovery domains into a shared domain where they can communicate with one another through uMiddle's common protocol. To achieve interoperability uMiddle makes use of mappers and translators. Mappers function as service-level and transport-level bridges. That is, they serve as bridges that connect service discovery (e.g. SLP) and binding (e.g. SOAP) protocols to uMiddle's common semantic space. Translators project service-specific semantics into the common semantic space, act as a proxy for that service and embody any protocol and semantics that are native to the associated service.

- The *Open Service Discovery Architecture (OSDA)* [20] is a scalable and programmable middleware for cross-domain discovery over wide-area networks (where a domain represents a particular discovery protocol. Its motivation is the need to integrate consumers and providers across different domains irrespective of the network they belong to. OSDA assumes that discovery agents (i.e. the service registry, service consumer and service provider) are already in place. To enable cross-domain service discovery, OSDA utilizes service brokers and a peer to peer indexing overlay. Service brokers function as interfaces between the OSDA inter-domain space and the different discovery systems and are responsible for handling and processing cross-domain service registrations and requests.
- *SeDiM* [21] is a component framework that self-configures its behaviour to match the interoperability requirements of deployed discovery protocols, i.e., if it detects SLP and UPnP in use, it creates a connector between the two. It can be deployed as either an interoperability platform (i.e. it presents an API to develop applications that will interoperate with all discovery protocols cf. ReMMoC), or it can be utilised as a transparent interoperability solution, i.e., it can be deployed in the infrastructure, or any available device in the network and it will translate discovery functions between the protocols in the environment. SeDiM provides a skeleton abstraction for implementing discovery protocols which can then be specialised with concrete middleware. These configurations can then be 'substituted' in an interoperability platform or utilised as one side of a bridge.

Transparent interoperability solutions allow interoperability to be achieved between two legacy-based platforms; and in this sense they meet the requirements for spontaneous interoperability. However, the fundamental problem with these approaches is the Greatest Common Divisor (GCD) problem; you must identify a subset of functionality between all protocols where they match. However, as the number of protocols increases this set becomes smaller and smaller restricting what is possible.

### 3.5 Logical Mobility

Logical mobility is characterised by mobile code being transferred from one device and executed on another. The approach to resolve interoperability is therefore straightforward; a service advertises its behaviour and also the code to interact with it. When a client discovers the service it will download this software and then use it. Note, such an approach relies on the code being useful somewhere, i.e., it could fit into a middleware as in the substitution approach, provide a library API for the application to call, or it could provide a complete application with GUI to be used by the user. The overall pattern is shown in Fig. 7. The use of logical mobility provides an elegant solution to the problem of heterogeneity; applications do not need to know in advance the implementation details of the services they will interoperate with, rather they simply use code that is dynamically available to them at run-time. However, there are fewer examples of systems that employ logical mobility to resolve interoperability because logical mobility is the weakest of the interoperability

approaches; it relies on all applications conforming to the common platform for executable software to be deployed. We now discuss two of these examples.

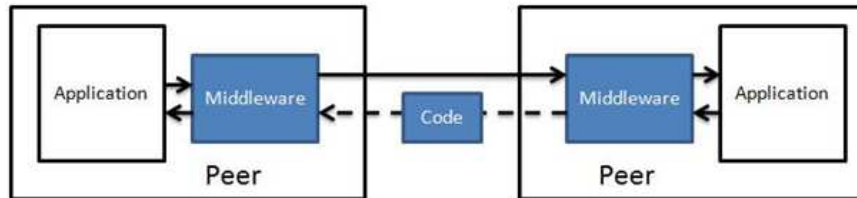


Fig. 7. Interoperability pattern utilised by Logical Mobility Solutions

**SATIN** [22] is a low footprint component based middleware that composes applications and the middleware itself into a set of deployable capabilities (a unit of functionality), for example, a discovery mechanism or compression algorithm. At the heart of SATIN is the ability to advertise and middleware capabilities. For example, a host uses SATIN to lookup the required application services; the interaction capabilities are then downloaded to allow the client to talk to the service.

**Jini** [23] is a Java based service discovery platform that provides an infrastructure for delivering services and creating spontaneous interactions between clients and services regardless of their hardware or software implementation. New services can be added to the network, old services removed and clients can discover available services all without external network administration. When an application discovers the required service, the service proxy is downloaded to their virtual machine so that it can then use this service. A proxy may take a number of forms: i) the proxy object may encapsulate the entire service (this strategy is useful for software services requiring no external resources); ii) the downloaded object is a Java RMI stub, for invoking methods on the remote service; and iii) the proxy uses a private communication protocol to interact with the service's functionality. Therefore, the Jini architecture allows applications to use services in the network without knowing anything about the wire protocol that the service uses or how the service is implemented.

## 4 Semantics-based Interoperability Solutions

### 4.1 Introduction

The previous middleware-based solutions support interoperation by abstract protocols and language specifications. But, by and large they ignore the data heterogeneity dimension. As highlighted in Section 2.1, for two parties to interoperate it is not enough to guarantee that the data flows across, but that they both build a semantic representation of the data that is consistent across the components boundaries. The data problem has been defined in Hammer and McLeod [24] as:

*“variations in the manner in which data is specified and structured in different components. Semantic heterogeneity is a natural consequence of*

*the independent creation and evolution of autonomous databases which are tailored to the requirements of the application system they serve”.*

Historically the problem has been well known in the database community where there is often the need to access information on different database which do not share the same data schema. More recently, with the advent of the open architectures, such as Web Services, the problem is to guarantee interoperability at all levels. We now look at semantics-based solutions to achieving interoperability: first, the Semantic Web Services efforts, second their application to middleware solutions, and third the database approaches.

#### 4.2 Semantic Web Services

The problem of data interoperability is crucial to address the problem of service composition since, for two services to work together, they need to share a consistent interpretation of the data that they exchange. To this extent a number of efforts, which are generically known as Semantic Web Services, attempt to enrich the Web Services description languages with a description of the semantics of the data exchanged in the input and output messages of the operations performed by services. The result of these efforts are a set of languages that describe both the orchestration of the services' operations, in the sense of the possible sequences of messages that the services can exchange as well as the meanings of these messages with respect to some reference ontology.

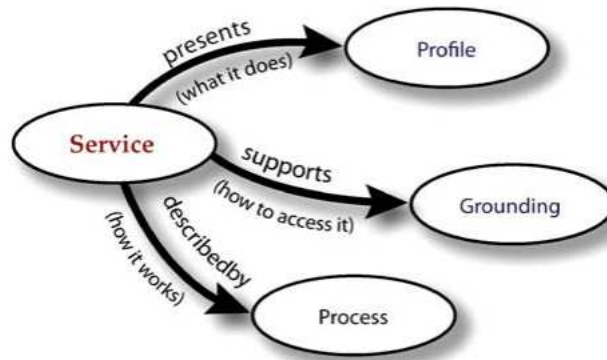


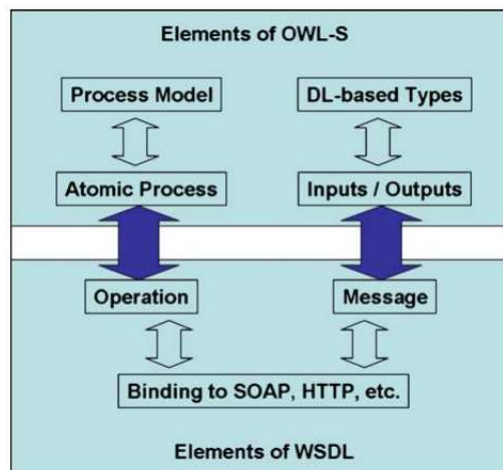
Fig. 8. OWL-S Upper Level Structure

**OWL-S** [26] and its predecessor DAML-S [25] have been the first efforts to exploit Semantic Web ontologies to enrich descriptions of services. The scope of OWL-S is quite broad, with the intention to support both service discovery through a representation of the capabilities of services, as well as service composition and invocation through a representation of the semantics of the operations and the messages of the service. As shown in Fig. 8, services in OWL-S are described at three different levels. The Profile describes the capabilities of the service in terms of the information transformation produced by the service, as well as the state

transformation that the service produces; the Process (Model) that describes the workflow of the operations performed by the service, as well as the semantics of these operations, and the Grounding that grounds the abstract process descriptions to the concrete operation descriptions in WSDL.

In more detail, the information transformation described in the Profile is represented by the set of inputs that the service expects and outputs that it is expected to produce, while the state transformation is represented by a set of conditions (preconditions) that need to hold for the service to execute correctly and the results that follow the execution of the service. For example, a credit card registration service may produce an information transformation that takes personal information as input, and returns the issued credit card number as output; while the state transformation may list a number of (pre)conditions that the requester needs to satisfy, and produce the effect that the requester is issued the credit card corresponding to the number reported in output.

The Process Model and Grounding relate more closely to the invocation of the service and therefore address more directly the problem of data interoperability. The description of processes in OWL-S is quite complicated, but in a nutshell they represent a transformation very similar to the transformation described by the Profile in the sense that they have inputs, outputs, preconditions and results that describe the information transformation as well as the state transformation which results from the execution of the process. Furthermore, processes are divided into two categories: *atomic processes* that describe atomic actions that the service can perform, and *composite processes* that describe the workflow control structure.



**Fig. 9.** The structure of the OWL-S process grounding

In turn atomic processes “ground” into WSDL operations as shown in Fig. 9 by mapping the abstract semantic descriptions of inputs and outputs of process into the WSDL message structures. In more detail, the grounding specifies which operations correspond to an atomic process, and how the abstract semantic representation is transformed in the input messages of the service or derived from the output messages. One important aspect of the Grounding is that it separates the OWL-S description of

the service from the actual implementation of the service, and therefore, every service which can be expressed in WSDL, can be represented in OWL-S. As a result of the service description provided by OWL-S the client service would always know how to derive the message semantics from the input/output messages of the service. Ideally therefore, the client may represent its own information at the semantic level, and then ground it to into the messages exchanged by the services.

**Analysis of OWL-S.** OWL-S provides a mechanism for addressing the data semantics; however it has failed in a number of aspects. First, many aspects of the service representation are problematic; for example, it is not clear what is the relation between the data representation of the atomic processes and the input/output representation of the complex (control flow) processes. Second, OWL-S is limited to a strict client/server model, as supported by WSDL, as a consequence it is quite unclear how OWL-S can be used to derive interoperability connectors in other types of systems. Third, OWL-S assumes the existence of an ontology that is shared between the client and server; this pushes the interoperability problem one level up. Of course the next data interoperability question is "what if there is not such a shared ontology?"

**SA-WSDL.** Semantic Web Services reached the standardization level with SA-WSDL [27], which defines a minimal semantic extension of WSDL. SA-WSDL builds on the WSDL distinction between the abstract description of the service, which includes the WSDL 2.0 attributes Element Declaration, Type Definition and Interface, and the concrete description that includes Binding and Service attributes which directly link to the protocol and the port of the service. The objective of SA-WSDL is to provide an annotation mechanism for abstract WSDL. To this extent it extends WSDL with new attributes:

1. *modelReference*, to specify the association between a WSDL or XML Schema component and a concept in some semantic model;
2. *liftingSchemaMapping* and *loweringSchemaMapping*, that are added to XML Schema element declarations and type definitions for specifying mappings between semantic data and XML.

The *modelReference* attribute has the goal of defining the semantic type of the WSDL attribute to which it applies; the lifting and lowering schema mappings have a role similar to the mappings in OWL-S since their goal is to map the abstract semantic to the concrete WSDL specification. For example, when applied to an input message, the model reference would provide the semantic type of the message, while the *loweringSchemaMapping* would describe how the ontological type is transformed into the input message.

A number of important design decisions were made with SA-WSDL to increase its applicability. First, rather than defining a language that spans across the different levels of the WS stack, the authors of SA-WSDL have limited their scope to augmenting WSDL, which considerably simplifies the task of providing a semantic representation of services (but also limits expressiveness). Specifically, there is no intention in SA-WSDL to support the orchestration of operations. Second, there is a deliberate lack of commitment to the use of OWL [28] as an ontology language or to any other particular semantic representation technology. Instead, SAWSDL provides a very general annotation mechanism that can be used to refer to any form of semantic markup. The annotation referents could be expressed in OWL, in UML, or in any

other suitable language. Third, an attempt has been made to maximize the use of available XML technology from XML schema, to XML scripts, to XPath, with the attempt to lower the entrance barrier to early adopters.

**Analysis of SA-WSDL.** Despite these design decisions that seem to suggest a sharp distinction from OWL-S, SA-WSDL shares features with OWL-S' WSDL grounding. In particular, both approaches provide semantic annotation attributes for WSDL, which are meant to be used in similar ways. It is therefore natural to expect that SAWSDL may facilitate the specification of the Grounding of OWL-S Web Services, a proposal in this direction has been put forward in [29]. The apparent simplicity of the approach is somewhat deceiving. First, SA-WSDL requires a solution to the two main problems of the semantic representation of Web Services: namely the generation and exploitation of ontologies, and the mapping between the ontology and the XML data that is transmitted through the wire. Both processes are very time consuming. Second, there is no obligation what-so-ever to define a modelReference or a schemaMapping for any of the attributes of the abstract WSDL, with the awkward result that it is possible to define the modelReference of a message but not how such model maps to the message, therefore it is impossible to map the abstract input description to the message to send to the service, or given the message of the service to derive its semantic representation. Conversely, when schemaMapping is given, but not the modelReference, the mapping is known but not the expected semantics of the message, with the result that it is very difficult to reason on the type of data to send or to expect from a service.

**Web Service Modelling Ontology (WSMO)** aims at providing a comprehensive framework for the representation and execution of services based on semantic information. Indeed, WSMO has been defined in conjunction with WSML (Web Service Modelling Language) [30], which provides the formal language for service representation, and WSMX (Web Service Modelling eXecution environment) [31] which provides a reference implementation for WSMO. WSMO adopts a very different approach to the modelling of Web Services than OWL-S and in general the rest of the WS community. Whereas the Web Service Representation Framework concentrates on the support of the different operations that can be done with Web Services, namely discovery with the Service Profile as well as UDDI [32], composition with the Process Model as well as BPEL4WS [33] and WS-CDL [34], and invocation with the Service Grounding, WSDL or SA-WSDL, WSMO provides a general framework for the representation of services that can be utilized to support the operations listed above, but more generally to reason about services and interoperability. To this extent it identifies four core elements:

- *Web Services*: which are the computational entities that provide access to the services. In turn their description needs to specify their capabilities, interfaces and internal mechanisms.
- *Goals*: that model the user view in the Web Service usage process.
- *Ontologies* provide the terminology used to describe Web Services and Goals in a machine processable way that allow other components and applications to take actual meaning into account.
- *Mediators*: that handle interoperability problems between different WSMO elements. We envision mediators as the core concept to resolve incompatibilities on the data, process and protocol level.

What is striking about WSMO with respect to the rest of the WS efforts (semantic and not) is the representation of goals and mediators as “first class citizens”. Both goals and mediators are represented as “by product” by the rest of the WS community. Specifically, in other efforts the users' goals are never specified, rather they are manifested through the requests that are provided to a service registry such as UDDI or to a service composition engine; on the other side mediators are either a type of service and therefore indistinguishable from other services, or generated on the fly through service composition to deal with interoperability problems. Ontologies are also an interesting concept in WSMO, because WSMO does not limit itself to use existing ontology languages, as in the case of OWL-S that is closely tied to OWL, nor it is completely agnostic as in the case of SA-WSDL. Rather WSMO relies on WSML which defines a family of ontological languages which are distinguished by logic assumptions and expressivity constraints. The result is that some WSML sub-languages are consistent (to some degree) with OWL, while others are inconsistent with OWL and relate instead to the DL family of logics.

Despite these differences, the description of Web Services has strong relations to other Web Services efforts. In this direction, WSMO grounds on the SA-WSDL effort (indeed SA-WSDL has been strongly supported by the WSMO initiative). Furthermore, the capabilities of a Web Service are defined by the state and information transformation produced by the execution of the Web Service, as was the case in OWL-S. The Interface of a Web Service is defined by providing a specification of its choreography which defines how to communicate with the Web Service in order to use its functions; and by the orchestration that reveals how the functionality of the service is achieved by the cooperation of more elementary Web Service providers. Of particular interest to addressing interoperability problems, WSMO defines three types of mediators:

1. *Data Level Mediation* - mediation between heterogeneous data sources, they are mainly concerned with ontology integration.
2. *Protocol Level Mediation* - mediation between heterogeneous communication protocols, they relate to choreographies of Web Services that ought to interact.
3. *Process Level Mediation* - mediation between heterogeneous business processes; this is concerned with mismatch handling on the business logic level of Web Services and they relate to the orchestration of Web Services.

**Analysis of WSMO.** WSMO put a strong emphasis on mediation and, as discussed above, it defines mediation as a “first class” citizen. The problem with WSMO is that that the WSMO project proposed an execution semantics for mediators [31] [35] but so far no theory or algorithm on how to construct mediators automatically has been proposed by the project. Somehow, it is curious that mediation is one of the fundamental elements of the approach while choreography is left to a secondary role within the specification of service definitions. Essentially it moves service composition to a secondary role in the theory.

### 4.3 Semantic Middleware

A number of research efforts have investigated middleware that support semantic specification of services for pervasive computing. These solutions mainly focus on providing middleware functionalities enabling semantic service discovery and composition as surveyed hereafter. **The Task Computing project** [36] is an effort for ontology-based dynamic service composition in pervasive computing environments. It relies on the UPnP service discovery protocol, enriched with semantic service descriptions given in OWL-S. Each user of the pervasive environment carries a service composition tool on his/her device that discovers on the fly available services in the user's vicinity and suggests to the user a set of possible compositions of these services. The user may then select the right composition among the suggested ones.

IGPF (**Integrated Global Pervasive Computing Framework**) [37] introduces a semantic Web Services-based middleware for pervasive computing. This middleware builds on top of the semantic Web paradigm to share knowledge between the heterogeneous devices that populate pervasive environments. The idea behind this framework is that information about the pervasive environments (i.e., context information) is stored in knowledge bases on the Web. This allows different pervasive environments to be semantically connected and to seamlessly pass user information (e.g., files/contact information), which allows users to receive relevant services. Based on these knowledge bases, the middleware supports the dynamic composition of pervasive services modelled as Web Services. These composite services are then shared across various pervasive environments via the Web.

The Ebiquty group describes a **semantic service discovery and composition protocol for pervasive computing**. The service discovery protocol, called GSD (Group-based Service Discovery) [38], groups service advertisements using an ontology of service functionalities. In this protocol, service advertisements are broadcasted to the network and cached by the networked nodes. Then, service discovery requests are selectively forwarded to some nodes of the network using the group information propagated with service advertisements. Based on the GSD service discovery protocol, the authors define a service composition functionality for infrastructure-less mobile environments [39]. Composition requests are sent to one of the composition managers of the environment, which performs a distributed discovery of the required component services.

The combined work in [40] and [41] introduces an efficient, **semantic, QoS-aware service-oriented middleware for pervasive computing**. The authors propose a semantic service model to support interoperability between existing semantic but also plain syntactic service description languages. The model further supports formal specification of service conversations as finite state automata, which enables automated reasoning about service behaviour independently of the underlying conversation specification language. Moreover, the model supports the specification of service non-functional properties to meet the specific requirements of pervasive applications. The authors further propose an efficient semantic service registry. This registry supports a set of conformance relations for matching both syntactic and rich semantic service descriptions, including non-functional properties. Conformance relations evaluate the semantic distance between service descriptions and rate services with respect to their suitability for a specific client request, so that selection can be

made among them. Additionally, the registry supports efficient reasoning on semantic service descriptions by semantically organizing such descriptions and minimizing recourse to ontology-based reasoning, which makes it applicable to highly interactive pervasive environments. Lastly, the authors propose flexible QoS-aware service composition towards the realization of user-centric tasks abstractly described on the user's handheld. Flexibility is enabled by a set of composition algorithms that may be run alternatively according to the current resource constraints of the user's device. These algorithms support integration of services with complex behaviours into tasks also specified with a complex behaviour; and this is done efficiently relying on efficient formal techniques. The algorithms further support the fulfilment of the QoS requirements of user tasks by aggregating the QoS provided by the composed networked services.

The above surveyed solutions are indicative of how ontologies have been integrated into middleware for describing semantics of services in pervasive environments. Semantics of services, users and the environment are put into semantic descriptions, matched for service discovery, and composed for achieving service compositions. Focus is mainly on functional properties, while non-functional ones have been less investigated. Then, efficiency is a key issue for the resource-constrained pervasive environments, as reasoning based on ontologies is costly in terms of computation.

#### 4.4 Beyond Web Services: DB Federation

The problem of data interoperation is by no means restricted to Web Services and middleware, rather it has been looked at the DB community for a long time. In this context, the data problem has been widely studied by the DB community while addressing the task of DB federation. Despite of the importance of the information stored in DBs, because of the way DBs and organizations evolve, the information stored on different databases is often very difficult to integrate. In this context "Database federation is one approach to data integration in which middleware, consisting of a relational database management system, provides uniform access to a number of heterogeneous data sources" [42]. Federated Data sources have a lot in common with the heterogeneous systems to be connected. They need to federate autonomous databases which are autonomously maintained, therefore they need to support a high degree of heterogeneity both at the architectural level, in the sense that they should host different version of databases made by different vendors as well support data heterogeneity because different nodes may follow different data schema.

The standard solution to the problem of data interoperability is to provide **Table User Defined Functions** (T-UDF) [42] which reformat the data from one database and present it in a format that is consistent with the format of a different data-base. For example, if one database provides address book information, a programmer may define a T-UDF `addressbook()` which reformats the data in the appropriate way, and then retrieve the data by using the SQL command `FROM TABLE addressbook()` in the query. T-UDF hardly provides a solution to the problem of data interoperability since they require a programmer that reformats the data from one data-schema to another.

Since the definition of translation functions as the T-UDF functions above is a very expensive process a considerable effort has been put into learning the translation between data-base schemata. Examples of these translations are provided in [43] [44]. They exploit a combination of machine learning, statistical processing and natural language lexical semantics to "guess" how two data-base schemata correspond. In Section 5.4 similar tools for ontology matching are analyzed more in detail.

The results of these mapping processes are mappings between data schemata that are correct up to a degree of confidence. The user should then find a way to deal with the reduced confidence in the results. One proposal in this direction has been provided by **Trio** [45], a data-base management system that extends the traditional data model to include data accuracy and lineage. Within Trio it is possible to express queries of the sort "find all values of X with approximation with confidence greater than K".

The approaches above ignore the most important information that is required for data mapping namely the explicit annotation of data semantics. Above, we discussed T-UDT as a mechanism for data translation mappings, but the problem with any form of mapping is that it makes assumptions on the semantics of the schemata that it is mapping across. There is therefore neither guarantee that these mappings are correct [46] nor that they will generalize if and when the schemata are modified. The automatic mapping mechanisms above, try to circumvent the problem of explicit semantics by using learning inference. But they assume semantics in the form of background knowledge such as lexical semantics without any guarantee that the background knowledge is relevant for the specific transformation. Essentially, the lack of explicit semantics emerges as an error in the accuracy of the transformation.

The development of ontologies, in the sense of shared data structures, is an alternative to the methods produced above. Essentially, instead of mapping all schemata directly in a hardcoded way as suggested by the T-UDT methods or try to guess the relation between schemata as suggested by the learning mechanisms, schemata are mapped to a unique "global" schema, indeed an ontology, from which direct mappings are derived. In this model the ontology provides the reference semantic for all schemata. The advantage of this model is that the DB provider could in principle provide the mapping to the ontology possibly removing the misinterpretation problem.

There are a number of problems of this approach. First, the ontology should be expressive enough to express all information within all the schemata in the federated databases. This implicitly requires a mechanism for extensible ontologies since adding new databases may require an extension of the ontology. Second, the derivation of mapping rules is proven to have an NP worst case computational complexity [47].

#### 4.4 Raising Interoperability one level up

The discussion about ontologies above immediately raises the question of whether and to what extent ontologies just push the interoperability problem somewhere else. Ultimately, what guarantees that the interoperability problems that we observe at the data structure level do not appear again at the ontology level? Suppose that different middlewares refer to different ontologies, how can they interoperate?

The ideal way to address this problem is to construct an *alignment ontology*, such as SUMO<sup>5</sup>, which provide a way to relate concepts in the different ontologies. Essentially, the alignment ontology provides a mapping that translates one ontology into the other. Of course, the creation of alignment ontologies not only requires efforts, but more importantly, it requires a commitment so that the aligning ontology is consistent with all ontologies to be aligned.

Such alignment ontologies, when possible, are very difficult to build and very expensive. To address this problem, in the context of the semantic web there is a very active subfield that goes under the label of *Ontology Matching* [48][49] which develops algorithms and heuristics to infer the relation between concepts in different ontologies. The result of an ontology matcher is a set of relations between concepts in different ontologies, and a level of confidence that that these relations hold. For example, an ontology matcher may infer that the concept *Price* in one ontology is equivalent to *Cost* in another ontology with a confidence of 0.95. In a sense, the confidence value assigned by the ontology matcher is a measure of the quality of the relations specified.

Ontology matching provides a way to address the problem of using different ontologies without pushing the data interoperability problem somewhere else. But this solution comes at a cost of the confidence on the on the interoperability solution adopted and ultimately on the overall system.

## 5 Analysis

The results of the state of the art investigation in Sections 3 and 4 shows two important things; first, there is a clear disconnect between the main stream middleware work and the work on application, data, and semantic interoperability; second, none of the current solutions addresses all of the requirements of dynamic pervasive systems as highlighted in the interoperability barriers in Section 2.

With respect to the first problem, it is clear that two different communities evolved independently. The first one, addressing the problems of middleware, has made a great deal of progress toward middleware that support sophisticated discovery and interaction between services and components. The second one, addressing the problem of semantic interoperability between services, however, inflexibly assuming Web Services as the underlying middleware; or the problem of semantic interoperability between data intensive components such as databases. The section on semantic middleware shows that ultimately the two communities are coming together, but a great deal of work is still required to merge the richness of the work performed on both sides.

---

<sup>5</sup> SUMO stands for: *Suggested Upper Merged Ontology*. It is available at: <http://www.ontologyportal.org/>

**Table 1.** Evaluation summary of effectiveness of interoperability solutions against each of the interoperability barriers

|                       | <b>SD = Discovery</b><br><b>I = Interaction</b><br><b>D= Data</b><br><b>A = Application</b><br><b>N=Non-functional</b> |          |          |          |          | <b>Transparency</b>   |
|-----------------------|--|----------|----------|----------|----------|---|
|                       | <b>SD</b>  | <b>I</b> | <b>D</b> | <b>A</b> | <b>N</b> |   |
| CORBA                 |  | X        |          |          |          | CORBA for all   |
| Web Services          |  | X        |          |          |          | WSDL & SOAP for all   |
| ReMMoC                | X  | X        |          |          |          | Client-side middleware  |
| UIC                   |  | X        |          |          |          | Client-side middleware  |
| WSIF                  |  | X        |          |          |          | Client-side middleware  |
| MDA                   |  | X        |          |          |          | Platform Independent models   |
| UniFrame              |  | X        |          |          |          | Platform Specific models  |
| ESB                   |  | X        |          |          |          | Bridge connector  |
| MUSDAC                | X  |          |          |          |          | Connection to middleware  |
| INDISS                | X  |          |          |          |          | Yes   |
| uMiddle               | X  | X        |          |          |          | Yes   |
| OSDA                  | X  |          |          |          |          | Yes   |
| SeDiM                 | X  |          |          |          | X        | Yes   |
| SATIN                 | X  | X        |          |          |          | Choice of SATIN for all   |
| Jini                  |  | X        |          |          |          | Choice of Jini for all  |
| Semantic Middleware   |  |          | X        | X        |          | Choice of same semantic middleware for all                          |
| Semantic Web Services |  | X        | X        | X        |          | WSDL for all plus commitment on a semantic framework and ontologies |

With respect to the second problem, namely addressing the interoperability barriers from Section 2 we pointed out that in such systems endpoints are required to spontaneously discover and interact with one another and therefore these three fundamental dimensions are used to evaluate the different solutions:

1. *Does the approach resolve (or attempt to resolve) differences between discovery protocols employed to advertise the heterogeneous systems? [Discovery column]*
2. *Does the approach resolve (or attempt to resolve) differences between interaction protocols employed to allow communication with a system? [Interaction column]*
3. *Does the approach resolve (or attempt to resolve) data differences between the heterogeneous systems? [Data column]*
4. *Does the approach resolve (or attempt to resolve) the differences in terms of application behaviour and operations? [Application column]*

5. *Does the approach resolve (or attempt to resolve) the differences in terms of non-functional properties of the heterogeneous system? [Non-functional column]*

The summary of this evaluation is in Table 1 (an x indicates: resolves or attempts to). This shows that no solution attempts to resolve all five dimensions of interoperability. Those that concentrate on application and data e.g. Semantic Web Services rely upon a common standard (WSDL) and conformance by all parties to use this with semantic technologies. Hence, transparent interoperability between dynamically communicating parties cannot be guaranteed. Semantic Web Services have a very broad scope, including discovery interaction and data interoperability, but these provide only a primitive support and languages to express the data dimension in the context of middleware solutions.

The transparency column shows that only the transparent interoperability solutions achieve interoperability transparency between all parties (however only for a subset of the dimensions). The other entries show the extent to which the application endpoint (client, server, peer, etc.) sees the interoperability solution. ReMMoC, UIC and WSIF rely on clients building the applications on top of the interoperability middleware; the remainder rely on all parties in the distributed system committing to a particular middleware or approach.

## 6 Conclusions and Future Work

This chapter has investigated the problem of interoperability in the complex distributed systems of today, with the added complexity stemming from the extreme level heterogeneity encountered in such systems coupled with the increasing level of dynamism of such systems which results in the need for spontaneous communication. The chapter highlights the key barriers to interoperability coupled with a discussion of solutions to interoperability featuring the research in the middleware community and related research on semantic interoperability. The most striking aspect of this study is that, while both communities focus on key interoperability problems, research efforts have to a large extent been disjoint. The other striking feature is that, despite considerable research efforts into interoperability dating back to the early 1980s, this remains a poorly understood area and currently solutions simply do not meet the needs on the complex distributed systems of today, particularly in terms of the levels of heterogeneity and dynamism as mentioned above.

The CONNECT project, an initiative funded under the Future and Emerging Technologies programme within the ICT theme of the European Commission's Framework programme, is taking a novel approach to the study of interoperability in complex distributed systems, going back to basics, and taking input from a variety of sub-disciplines including the middleware and semantic web communities, but also looking at supportive areas such as formal semantics of distributed systems, learning and synthesis technologies and support for dependable distributed systems. We propose an approach that:

- places semantic understanding of concepts at the heart of achieving interoperability,

- seeks a dynamic approach to interoperability where appropriate infrastructure is generated on-the-fly for the current context (emergent middleware), and this involves enabling technologies such as learning and synthesis of run-time connectors,
- grounds itself in formal semantics enabling validation and verification to be carried out,
- addresses the dependability requirements of modern distributed systems, including meeting the associated non-functional requirements in highly heterogeneous environments,
- supports dynamism allowing currently deployed solutions to be constantly monitored and adapted to changing context.

The rest of the book unfolds this story in more detail with chapter 2 providing an overview of the Connect architecture and other chapters unfolding key enabling technologies behind this approach.

## References

1. Bouquet, P., Stoermer, H., Nederee, C., Mana, A.: Entity Name System: The Backbone of an Open and Scalable Web of Data. In: Proceedings of the IEEE International Conference on Semantic Computing (ICSC 2008), pp. 554-561 (2008).
2. Van Steen, M., Tanenbaum, A.: Distributed Systems: Principles and Paradigms. Prentice-Hall (2001)
3. Object Management Group.: The common object request broker: Architecture and specification Version 2.0. OMG Technical Report (1995)
4. Microsoft Corporation.: Distributed Component Object Model (DCOM) Remote Protocol Specification.  
<http://msdn.microsoft.com/en-gb/library/cc201989%28PROT.10%29.aspx>
5. Srinivasan, R.: RPC: Remote Procedure Call Protocol Specification Version 2. Network Working Group RFC1831, <http://tools.ietf.org/html/rfc1831> (1995)
6. Microsoft Corporation.: Microsoft Message Queuing.  
<http://www.microsoft.com/windowsserver2003/technologies/msmq/>
7. Carzaniga, A., Rosenblum, D., Wolf, A.: Design and Evaluation of a Wide-Area Event Notification Service. ACM Transactions on Computer Systems, 19:3, 332-383 (2001)
8. Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Language and Systems, 7:1, 80-112 (1985)
9. Wyckoff, P., McLaughry, S., Lehman, T., Ford, D: Tspaces. IBM Systems Journal, 37:3, 454-474 (1998)
10. Davies, N., Friday, A., Wade, S., Blair, G.: L<sup>2</sup>imbo: A Distributed Systems Platform for Mobile Computing. ACM Mobile Networks and Applications (MONET), 3:2, 143-156 (1998)
11. Murphy, A., Picco, G., Roman, G.: LIME: A Middleware for logical and Physical Mobility. In: 21st International Conference on Distributed Computing Systems (ICDCS-21), pp. 524-533 (2001)
12. Booth D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture. W3C Working Group Note, <http://www.w3.org/TR/ws-arch/> (2004)
13. Roman, M., Kon, F., Campbell, R.: Reflective Middleware: From Your Desk to Your Hand. IEEE Distributed Systems Online, 2:5 (2001)

14. Kon F., Román, M., Liu, P., Mao, J., Yamane, T., Magalhães, L., Campbell, R.: Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In: Proceedings of the 2nd International ACM/IFIP Middleware Conference, pp. 121-143 (2000).
15. Grace, P., Blair, G., Samuel, S.: A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments. ACM SIGMOBILE Mobile Computing and Communications Review, 9:1, 2-14 (2005)
16. Duftler, M., Mukhi, N., Slominski, S., Weerawarana, S.: Web Services Invocation Framework (WSIF). In: Proceedings of OOPSLA 2001 Workshop on Object Oriented Web Services, Tampa, Florida (2001).
17. Object Management Group.: COM/CORBA Interworking Specification Part A & B. OMG Technical Report orbos/97-09-07 (1997)
18. Bromberg, Y., Issarny, V.: INDISS: Interoperable Discovery System for Networked Services. In: Proceedings of the IFIP/ACM/Usenix International Middleware Conference, Grenoble, France, pp. 164-183 (2005)
19. Nakazawa, J., Tokuda, H., Edwards, W., Ramachandran, U.: A Bridging Framework for Universal Interoperability in Pervasive Systems. In: Proceedings of 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006), Lisbon, Portuga, (2006)
20. Limam N., Ziemicki, J., Ahmed, R., Iraqi, Y., Li, D., Boutaba, R., Cuervo, F.: OSDA: Open service discovery architecture for efficient cross-domain service provisioning. Computer Communications, 30:3, 546-563 (2007)
21. Flores, C., Grace, P., Blair, G.: SeDiM: A Middleware Framework for Interoperable Service Discovery in Heterogeneous Networks. ACM Transactions on Autonomous and Adaptive Systems, 6:1, article 6 (2011).
22. Zachariadis, S., Mascolo, C., Emmerich, W.: Satin: A Component Model for Mobile Self-Organisation, In: Meersman, R. and Tari, Z., (eds.) On the Move to Meaningful Internet Systems 2004: Proceedings of CoopIS, DOA and ODBASE, Agia Napa, Cyprus, pp. 1303-1321. Springer Verlag (2004)
23. Arnold, K., O'Sullivan, B., Scheifler, R., Waldo, J., Wollrath, A.: The Jini Specification. Addison Wesley (1999)
24. Hammer, J., McLeod, D.: An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. Int. J. Cooperative Inf. Syst, 2:1, 51-83 (1993)
25. Burstein M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K.: DAML-S: Web Service Description for the Semantic Web. In: International Semantic Web Conference, pp. 348-363 (2002)
26. Martin D., Burstein, M., Mcdermott, D., Mcilraith, S., Paolucci, M., Sycara, K., Mcguinness, D., Sirin, E., Srinivasan, N.: Bringing Semantics to Web Services: The OWL-S Approach. In: First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), pp. 26-42 (2004)
27. Farrell J., Lausen, H.: Semantic Annotations for WSDL and XML Schema. W3C Recommendation. <http://www.w3.org/TR/sawSDL/> (2007)
28. McGuinness D., Harmelen, F.: OWL Web Ontology Language. W3C recommendation. <http://www.w3.org/TR/owl-features/> (2004)
29. Martin, D., Paolucci, M., Wagner, M.: Bringing Semantic Annotations to Web Services: OWL-S from the SAWSDL Perspective. In: 6th International and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007), pp. 340-352 (2007)
30. de Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L.: The Web Service Modeling Language WSMML. <http://www.wsmo.org/TR/d16/d16.1/v0.21/> (2005)

31. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - a semantic service-oriented architecture. In: Proceedings of the International Conference on Web Services (ICWS 2005), Orlando, Florida, pp. 321- 328 (2005).
32. OASIS: Univeral Description, Discovery and Integration of Web Services. <http://www.uddi.org> (2002)
33. Jordan D., Evdemon, J.: Web Services Business Process Execution Language (WSBPEL) Version 2.0. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html> (2007)
34. Kavantzias N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., Barreto, C.: Web Services Choreography Description Language Version 1.0. <http://www.w3.org/TR/ws-cdl-10/> (2005)
35. Cimpian E., Mocan, A.: WSMX Process Mediation Based on Choreographies. In: 1<sup>st</sup> International Workshop on Web Service Choreography and Orchestration for Business Process Management, Nancy, France (2005)
36. Masuoka, R., Parsia, B., Labrou, Y.: Task Computing – the Semantic Web Meets Pervasive Computing. In: Proceedings of the 2nd International Semantic Web Conference (ISWC2003) (2003)
37. Singh, S., Puradkar, S., Lee, Y.: Ubiquitous Computing: Connecting Pervasive Computing Through Semantic Web. *Information Systems and e-Business Management Journal*, 4:4, 421-439 (2005)
38. Chakraborty, D., Joshi, A., Finin, T.: Toward Distributed Service Discovery in Pervasive Computing Environments. *IEEE Transactions on Mobile Computing*, 5:2, 97–112 (2006)
39. Chakraborty, D., Joshi, A., Finin, T., Yesha, Y.: Service Composition for Mobile Environments. *Journal on Mobile Networking and Applications, Special Issue on Mobile Services*, 10: 4, 435-451 (2005)
40. Ben Mokhtar, S., Georgantas, N., Issarny, V.: COCOA: CONversation-based Service Composition in Pervasive Computing Environments with QoS Support. *Journal of Systems and Software, Special Issue on ICPS'06*, 80:12, 1941–1955 (2007).
41. Ben Mokhtar, S., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y.: EASY: Efficient SemAntic Service DiscoverY in Pervasive Computing Environments with QoS and Context Support. *Journal of Systems and Software, Special Issue on Web Services Modelling and Testing*, 81:5, 785-808 (2008)
42. Haas, M., Lin, E., Roth, M.: Data integration through database federation. *IBM Systems Journal*, 41:4, 578-596 (2002)
43. Jung, J.: Taxonomy alignment for interoperability between heterogeneous virtual organizations. *Expert Systems with Applications*, 34:4, 2721–2731 (2008)
44. Berlin, J., Motro, A.: Database schema matching using machine learning with feature selection. In: Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE '02), pp. 452-466 Springer-Verlag, London, UK (2002)
45. Widom, J.: Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In: Second Biennial Conference on Innovative Data Systems Research (CIDR '05), Pacific Grove, California (2005)
46. Vetere, G., Lenzerini, M.: Models for semantic interoperability in service-oriented architectures. *IBM Systems Journal*, 44:4, 887-904 (2005)
47. Fagin, P., Kolaitis, P., Popa, L.: "Data Exchange, Getting to the Core. In: Symposium of Principles of Database Systems, pp. 90-101, ACM, New York (2003)
48. Euzena, J. and Shvaiko, P. *Ontology matching*. Springer-Verlag. 2007
49. Shvaiko, P., Euzenat J., Giunchiglia F., Stuckenschmidt H., Mao, M. Cruz, I. Proceedings of the 5th International Workshop on Ontology Matching (OM-2010). CEUR. 2010