Initial CONNECT Architecture

Antonia Bertolino, Gordon Blair, Franck Chauvel, Carlos Flores Cortes, Nikolaos Georgantas, Paul Grace, Falk Howar, Tran Huyn, Bengt Jonsson, Massimo Paolucci, et al.

CONNECT

Emergent Connectors for

Eternal Software Intensive Networked Systems

ICT FET IP Project

Deliverable D1.1

Initial CONNECT Architecture

SEVENTH FRAMEWORK PROGRAMME

http://www.connect-forever.eu

INRIA docomo DOCOMO Euro-Labs LANCASTER UNIVERSITY THALES Universita' dell'Aquila technische universität dortmund UNIVERSITY OF OXFORD UPPSALA UNIVERSITET

Project Number	:	231167
Project Title	:	CONNECT – Emergent Connectors for Eternal Software Intensive Networked Systems
Deliverable Type	:	Report

Deliverable Number	:	D1.1
Title of Deliverable	:	Initial CONNECT Architecture
Nature of Deliverable	:	R
Dissemination level	:	Public
Internal Document Number	:	24
Contractual Delivery Date	:	February 1st 2010
Actual Delivery Date	:	February 15th 2010
Contributing WPs	:	WP1
Editor(s)	:	Paul Grace (LANCS), Gordon Blair (LANCS)
Author(s)	:	Antonia Bertolino (CNR), Gordon Blair (LANCS), Franck Chauvel (PKU), Carlos Flores Cortes (LANCS), Nikolaos Georgantas (INRIA), Paul Grace (LANCS), Falk Howar (TUDO), Tran Huyn (TCF), Bengt Jonsson (UU), Massimo Paolucci (DOCOMO), Animesh Pathak (INRIA), Bertrand Souville (DOCOMO), Massimo Tivoli (UNIVAQ)
Reviewer(s)	:	Romina Spalazzese (UNIVAQ), Antonino Sabetta (CNR), Amel Bennaceur (INRIA)

Abstract

Interoperability remains a fundamental challenge when connecting heterogeneous systems which encounter and spontaneously communicate with one another in pervasive computing environments. This challenge is exasperated by the highly heterogeneous technologies employed by each of the interacting parties, i.e., in terms of hardware, operating system, middleware protocols, and application protocols. The key aim of the CONNECT project is to drop this heterogeneity barrier and achieve universal interoperability. Here we report on the development of the overall CONNECT architecture that will underpin this solution; in this respect, we present the following contributions: i) an elicitation of interoperability requirements from a set of pervasive computing scenarios, ii) a survey of existing solutions to interoperability, iii) an initial view of the CONNECT architecture, and iv) a series of experiments to provide initial validation of the architecture.

Keyword list

Interoperability, middleware, middleware heterogeneity, service discovery heterogeneity, service interaction heterogeneity, application-level heterogeneity, data heterogeneity, non-functional properties, software architecture, connectors, semantics, ontologies

Document History

Version	Type of change	Author(s)
1	First Version	Grace
2	Ontology Matching Experiments	Souville
3	Third version	Chauvel
4	Fourth Version	Grace
5	Semantic Interoperability	Paolucci
6	Architecture	Georgantas
7	Bibliography	Paolucci
8	Complete Architecture	Georgantas
9	Semantic Middleware	Georgantas
10	V1 for reviewers	Grace
11	Improvements Section 4	Georgantas
12	V2 for reviewers	Grace
13	Answer to reviewers	Grace
14	Minor Fixes	Grace/ Georgantas
15	Minor Fixes	Pathak
14	Added second semantic experiment	Paolucci
16	Minor Fixes	Grace
17	Addressing Antonino issue	Georgantas
18	Revisions that Animesh sent	Grace
19	Connect/CONNECT	Grace
20	Added keywords (all). Minor fixes to Sections 3.6 and 5	Georgantas
21	Answer to Romina's comment	Souville
22	Minor fixes, changes to 5.1 workflow	Grace/Pathak
23	Changes to section 5.1	Pathak
24	Final	Pathak/Grace

Document Review

Date	Version	Reviewer	Comment
<date of review>	<version >	<name & affiliation>	<proofed, found weaknesses, corrections …>
26/01/2010	12	Romina Spalazzese, UNIVAQ	Proofed. List of corrections, and requests for clarification.
29/01/2010	12	Antonino Sabetta, CNR	Request for greater input about ontologies. Clarifications about role of monitoring in architecture. List of corrections.
01/02/2010	12	Amel Bennaceur, INRIA	Proofed. List of corrections. Request for consistency of interoperability dimensions used through document. Request for consistent analysis approach for Section 3.
06/02/2010	15	Valerie Issarny	Proofed. List of corrections. Suggestions for improvements to section 5.1.

Table of Contents

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, MANET applications, Enterprise systems, and sensor networks. Instead, these technology-dependent islands are themselves dynamically composed and connected together to create richer, pervasively deployed systems. While there are many challenges to obtaining this overall goal, the fundamental one is 'interoperability' i.e., the ability of one or more systems to: connect, understand and exchange data with one another. When considering interoperability there are two key properties:

- *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/spontaneous Communication*. Connections between systems are not made until runtime; no design or deployment decision, e.g., choice of middleware, can inform the interoperability solution.

With such characteristics, using common middleware technologies (with or without common standards) is unsuitable in practice, as they themselves add to the interoperability problem; namely, middleware A does not interoperate with middleware B. The CONNECT project aims to drop this heterogeneity barrier and enable the continuous composition of networked systems independently of the embedded technologies. Rather than create a middleware solution destined to be yet another legacy platform, CONNECT proposes *emergent middleware,* i.e., where such middleware provides runtime interoperability between two systems that spontaneously interact on the fly. CONNECT synthesizes CONNECTors that resolve interoperability at the data (e.g., heterogeneous data formats), application protocol (for example, different instant messaging or printing protocols) and middleware protocol (e.g., different service discovery or RPC protocols) layers.

The original three tasks of WP1 as described in the description of work [1] are as follows:

> ***Task 1.1: CONNECT architecture.*** *Elaborating a technology-independent and eternal architectural framework for emergent connectors or CONNECTors.*
>
> ***Task 1.2: Eternal system semantics.*** *Eliciting an ontology-based characterization of the semantics of connected systems.*
>
> ***Task 1.3: CONNECT realization.*** *Developing key underlying systems principles and techniques to support the development of a practical, efficient and a self-sustaining CONNECT prototype.*

The objective of this report is to provide an initial description of the CONNECT architecture. For this purpose, we present a number of contributions with respect to the original three tasks:

- An analysis of typical complex pervasive systems is given in Section 2; this highlights the particular interoperability challenges in terms of discovery, interaction and data protocols employed.
- An investigation of the state of the art in middleware and data interoperability is provided in Section 3, which shows that no current solution can achieve the interoperability proposed by CONNECT; and hence CONNECT can make significant contributions to the field.
- A description of the initial common CONNECT architecture and architectural principles to underpin the creation and deployment of emergent CONNECTors is presented in Section 4. This will be refined as the project progresses by taking inputs from the other work

packages that focus on specific aspects of the architecture, and from the experience gained in applying the CONNECT concepts.

- Section 5 offers a validation of the initial architecture in terms of a number of small experiments that highlight the effectiveness of integrating the separate work package contributions, and the specific technological challenges to be resolved in the future.
- Identification of the importance of ontologies in the role of achieving interoperability between heterogeneous networked systems. We present a state of the art in data interoperability approaches and semantic middleware in Section 3.6; we then investigate initial ontology-based solutions within the CONNECT architecture in Sections 5.3 and 5.4.

2 Summary of Scenarios and Interoperability Challenges

In the initial phase of work, we examined scenarios emanating from each project partner to understand the requirements for interoperable middleware. A complete list of these scenarios can be found in the D6.1 deliverable. Our strategy was to take a broad look at all scenarios then zoom in on one and examine this in depth. Hence, in this section we investigate the distributed marketplace application scenario and illustrate the challenges that a CONNECTed system is expected to address and overcome.

2.1 Context: The Distributed Marketplace Scenario

As detailed in Sections 2.3 to 2.5 of deliverable D6.1, the overall setting of this scenario is a stadium where fans from various countries have gathered together to watch a game. The specific application we focus on in this section is that of a distributed marketplace (see Figure 2.1), as introduced in Section 2.5 of D6.1. Here, merchants publicise their wares, and consumers can search the market, and order from a merchant.

Figure 2.1: Distributed Marketplace

To properly simulate a realistic situation in the CONNECT world, we assume that there already exist various implementations of the distributed marketplace application, implemented by various developers in their own countries. However, these are not interoperable at the outset, due to the choices made during the design and development phases of these applications.

2.1.1 Common User Interface Requirements

In accordance with the scope of CONNECT, we assume that each implementation shares a common user behaviour, thus rendering these various implementations *"possibly interoperable"*. Specifically, here are the actions from the point of view of the Merchant and the Consumer:

Merchant: First, he publishes info on his product, which the consumers can browse through. When a consumer requests a product, he gets a notification of the amount ordered and the location of the consumer, to which he can respond with a yes/no. If yes, then when he is close enough to the user, both he and the consumer get a proximity notification by means of their

mobile device (ring/buzz). Periodically, the merchant can go back to his depot, and load up more of the product.

Consumer: Upon entering the stadium, the consumer can browse all products available in the market, and then see a listing of all merchants selling a particular product in a given vicinity with at least a certain amount to offer (e.g., "I want 3 bags of popcorn within 200m"). Then he can choose to submit a request to a particular merchant, receive a response, and wait at his seat until the merchant arrives and he gets a proximity notification on his mobile device.

2.1.2 Assumptions

For this scenario we make the following assumptions:

1. Consumers are stationary
2. Upon receiving a request for purchase from a consumer, the merchant cannot service other consumers until he either responds with a “no”, or responds with a “yes” and then delivers the merchandize.
3. Payment is NOT part of the system.

2.2 Interoperability Challenges

The setup of the distributed marketplace application allows us to explore several dimensions of interoperability challenges. In this section, we discuss the specific options available to the developer in each of these dimensions of heterogeneity.

2.2.1 Platform Level Heterogeneity

Applications can be deployed upon a number of platforms; the enterprise systems in each stadium and the mobile phone devices can each use a range of Operating Systems (Windows, Linux, Android, Windows Mobile, etc.) or virtual machine technologies (e.g. Sun’s Java JVM, or Microsoft’s .NET platform) and also be written in diverse programming languages (ranging from C to Ruby).

Although the choice of the platform itself might not be a large problem in networked applications, it might limit the developer to choosing certain middleware implementations.

2.2.2 Middleware Level Heterogeneity

At the middleware layer, we focus on two distinct interaction patterns available to the developer.

1. **Tuple Spaces:** In this methodology, the application components communicate by reading and writing *tuples* onto a shared space [2]. Each tuple is an ordered set (e_1, e_2, …, e_n), which can be *written* to or *read* from the tuple space. The tuple space also provides powerful search primitives, similar to relational databases. Additionally, the LIME (Linda for Mobile Environments) tuple space Java implementation, which we consider for the scenario, provides primitives for applications to submit requests to the tuple space, and then get notified when a tuple matching the description in the request is written.
2. **Service Discovery followed by Message Passing:** In this methodology, the consumer has to first discover the presence of, and the services provided by, the merchants. After that, the consumer can send direct messages to the merchants, and receive responses to them, one method for this step being the SOAP protocol specification [3]. We further explore two variations of this interaction pattern:

a. **Active Service Discovery:** This is achieved by way of a centralized directory, where the merchant can register with its information. Some implementations of the Service Location Protocol (SLP) [4] behave this way.

b. **Passive Service Discovery:** In this case, the consumer uses IP-based multicast to send discovery requests to the merchants, who then respond directly to the consumer with information. The Simple Service Discovery Protocol (SSDP) [5] of Universal Plug-and-Play (UPnP) [6] protocol stack uses this technique.

2.2.3 Application Level 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 in our scenarios, we assume that in approaches using service discovery, the merchant provides two methods for the consumer to obtain information about his wares.

1. A single `GetInfo()` remote method, which returns all the information about the product needed by the consumer, or
2. Three separate remote methods `GetLocation()`, `GetPrice()`, and `GetQuantity()`, using which the consumer can get to know parts of the information available about the product the merchant is selling.

The developer can code the consumer using either one of the manners 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. E.g, when using a Tuple Space, the programmer can use the rich search semantics provided by it, which is not possible while using a discovery protocol such as SSDP.

2.2.4 Data-representation Heterogeneity

A final set of interoperability challenges comes from the fact that different implementations may choose to represent data in different ways. In our scenarios, we will consider the cases where the different developers assume different currencies for the price of the goods (for example in the stadium scenario in Section 2.3, Euros are used in French and German stadiums, Pound Sterling in British stadiums, and Swiss Francs in Swiss stadiums).

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. Continuing on the example above, the SOAP client may represent the price of the popcorn using XML, while the tuple space may serialize its data using a Java-like syntax. So the simple information that the pop-corn cost € 1 may result in the two very different representations shown in Figure 2.2 below.

<price> <value> 1 </value> <currency> euro </currency> </price>	price(1,euro)

Figure 2.2: Representing price in XML and tuple data

Aside from the syntactic level interoperability due to the format of the data, there is a greater problem with the "meaning" of the tokens in the messages. Even if the two components used 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 examples in Figure 2.3 below.

<price> <value> 1 </value> <currency> euro </currency> </price>	<cost> <amount> 1 </ amount > <denomination> €</ denomination > </cost>

Figure 2.3: Heterogeneous Currency Data

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 it will fail to recognize the similarity between them unless it realizes that *price*≡*cost*, that *value*≡*amount*, that *currency*≡*denomination* and of course that *euro*≡*€ (where* ≡ *means equivalent).* The net result of using XML is that both systems will be in the awkward situation of parsing each other message, but not know what to do with the information that they just received.

The deeper problem of data heterogeneity is the semantic interoperability 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 [7], 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. In general, two systems may fragment data in different ways, the examples of the different calls described in Section 2.2.3 where the call `GetInfo()` may be equivalent to the result of three different calls: `GetLocation()`, `GetPrice()`, and `GetQuantity()`shows that ultimately `info`≈`location + price + quantity` (where ≈ is approximation), or, in other words, it should be recognized that the information provided by `GetInfo()` is the result of the aggregation of the information provided by the other three calls.

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.

2.3 List of Possible Configurations of Distributed Marketplace

Using the heterogeneity dimensions discussed above, we propose in Table 2.1, a set of different implementations of the distributed marketplace. Each one is identified by a label with a country name. We are assuming that all the stadia of that country natively support the platform, middleware, application, and data currency specified in Table 2.1. Trying to interoperate between merchants and consumers from any two (or more) of these implementations will lead to a scenario where CONNECT will be applicable to solve interoperability challenges. A set of possible scenarios are listed in Table 2.1.

Table 2.1: List of Implementation Choices for Distributed Marketplace

No.	Country	Platform	Middleware	Application	Data/ Currency
1	Germany	Java	Tuple Space	`GetInfo`	EUR
2	Great Britain	Microsoft	SLP+SOAP	`GetInfo`	GBP
3	France	Java	SSDP+SOAP	`GetInfo`	EUR
4	Italy	Java	SSDP+SOAP	`GetLocation+ GetPrice+ GetQuantity`	EUR
5	Switzerland	Java	SSDP+SOAP	`GetInfo`	CHF
6	Spain	Java	SLP+SOAP	`GetInfo`	EUR

To test the functionalities of the various enablers in CONNECT, one can choose pairs of implementations that suffer from only one dimension of heterogeneity. E.g, The German and French implementations only differ at the middleware level, while the French and Swiss implementations differ only at the data level.

2.4 Summary

The distributed marketplace scenario, and the other scenarios presented in deliverable D6.1, illustrate that the following important heterogeneity characteristics are typically found in dynamic pervasive computing systems and are limiting factors in achieving interoperability:

- *Platform Heterogeneity.* Applications are developed upon a wide range of devices and operating systems.
- *Middleware Heterogeneity.* Applications use a range of heterogeneous middleware protocols i.e. in terms of discovering and interacting with services.
- *Application-level Heterogeneity.* Applications have heterogeneous interfaces in terms of the descriptions of operations (e.g. a composed getInfo() operation, versus separate getPrice() and getDescription() operations), and they are also heterogeneous in terms of the order in which operations must/should be called.
- *Non-functional properties.* Peers may have particular non-functional properties e.g. latency of message delivery, dependability measures and security requirements that must be resolved with respect to the dynamically connected peer.
- *Data Heterogeneity.* Applications may use data that is represented in different ways and/or have different meanings.

Platform heterogeneity has well-known solutions provided by middleware technologies and will not be further addressed by the CONNECT project. However, the problems of middleware, application, non-functional and data heterogeneity remain challenging. We show in the subsequent chapter that the state of the art in middleware does not fully address these challenges with respect to complex pervasive computing applications.

3 State of the Art Interoperability Solutions

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"* [8].

Achieving such interoperability between independently developed systems has been one of the fundamental goals of middleware researchers and developers; and prior efforts have largely concentrated on solutions where conformance to one or other standard is required e.g. as illustrated by the significant standards work produced by the OMG for CORBA middleware [9], and by the W3C for Web Services based middleware [10], [11]. Such solutions have been successful in connecting systems developed on different hardware platforms, operating systems and in different programming languages. Indeed, if a single standard for the development of distributed systems had been agreed upon the problem of interoperability would have been solved. Unfortunately, this is not the case due to the heterogeneity of middleware solutions themselves. While applications developed upon the same platform can interoperate with one another, applications developed on different middleware platforms cannot. For example RMI protocols such as SOAP and RMI cannot interoperate; and all RMI protocols cannot interoperate with all tuple-space or message-based protocols. Hence, *middleware heterogeneity* is the next challenge middleware designers must address.

To tackle the problem of middleware-based heterogeneity there are five important interoperability problems that are a richer dissection of the heterogeneity identified in Section 2.4. Middleware heterogeneity now includes the different types of middleware protocols, e.g., discovery protocols and interaction protocols; and platform heterogeneity is assumed to be resolved by the state of the art. The interoperability dimensions are:

- *Discovery protocol interoperability*. It should be possible for services to advertise to, and find one another irrespective of the discovery protocol they themselves employ.
- *Interaction protocol interoperability*. Two or more services whose interaction protocol (e.g. RMI, messaging, etc.) differ can be bound together in order to interoperate.
- *Data interoperability*. The application data of the services must be semantically equivalent between the two parties, and transformed to a format that the receiver can understand and process (after the binding between the heterogeneous protocols has been created).
- *Application Interoperability*. Differences between application interfaces can be resolved.
- *Interoperability of Non-functional properties.* Interoperability can be achieved between systems while maintain the non-functional properties of each.

In this chapter we examine the state of the art in solutions to interoperability, and in particular, we evaluate their effectiveness in overcoming the five dimensions of interoperability described above.

In Section 3.1 we first look at typical standards-based platforms and middleware styles (e.g. RPC, event, tuple-space); second (in Sections 3.2 to 3.6), we discuss the research led platforms that have begun to address the middleware heterogeneity problem identified above in Section 2.2.2; and third, in Section 3.6, we investigate solutions to semantic and data interoperability issues. Finally, in Section 3.7, we summarize this state-of-the-art and show that no current solutions fully explore and address all of the interoperability dimensions.

3.1 Traditional Middleware

In this section, we give a brief overview of the traditional range of middleware solutions available to the developers of distributed systems. 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. However, all of the solutions follow a common design approach, where all parties in the distributed system implement their application upon the same middleware implementation or standard; this pattern is illustrated in Figure 3.1. 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 would be required to be implemented upon the same middleware). One-size fits all middleware standards have been attempted i.e. CORBA and Web Services, and have failed to achieve standardisation (even within the enterprise domain, and not considering the diversity of pervasive computing systems); we discuss these attempts in the following two sections and analyse why they failed in Section 3.2.4.

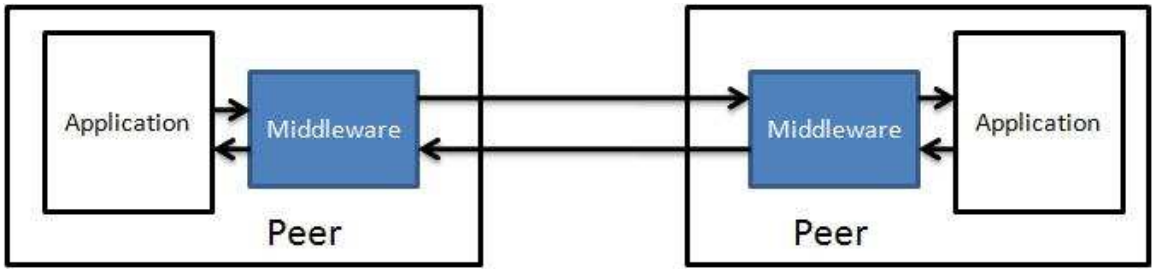


Figure 3.1: Traditional middleware-based Interoperability

3.1.1 Middleware Styles

There are many different middleware' styles that themselves contribute to the interoperability problem i.e. different styles do not interoperate; to highlight this point the following is a list of the most commonly used solution types (n.b. this is not an exhaustive list; provide further studies of middleware styles):

- *RPC/Distributed Objects.* Distributed Objects (e.g. CORBA [9] and DCOM [12]) 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 [13].
- *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 [14] and Java JMS [15].
- *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 [16] and JMS [15].

- *Tuple Spaces*. The Linda platform [2] 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 T-Spaces [17] and JavaSpaces [18] use a central enterprise server to host the tuple space for clients to connect to, while L^2imbo [19] and LIME [20] distribute the tuple space evenly among the peers.

3.1.2 Common Object Request Broker Architecture (CORBA)

The Object Management Group (OMG) defined a standard distributed open systems framework named the Common Object Request Broker Architecture (CORBA) to address the problems of developing portable distributed applications for heterogeneous systems. The fundamental component of CORBA is the ORB (illustrated in Figure 3.2); this allows clients to transparently invoke the operations of objects hosted remotely. The low level mechanism is a synchronous Remote Procedure Call (RPC).

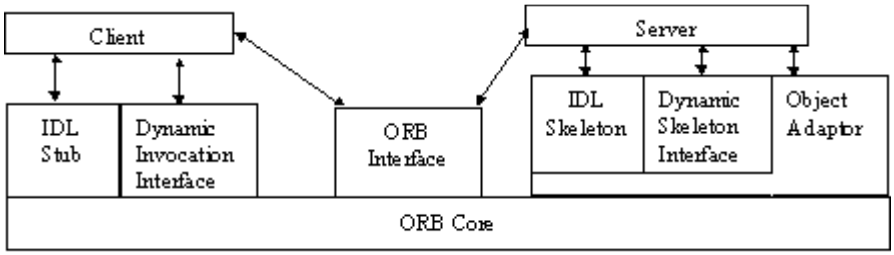


Figure 3.2: The Object Request Broker

CORBA utilises a number of key techniques to address interoperability solutions:

- Object interfaces are defined in a common IDL (Interface Definition Language), which provides a language independent method to define the structured data types and operation signatures clients can communicate through.
- The General Inter-ORB Protocol (GIOP) specification defines the Common Data Representation (CDR) for encoding method calls and the message formats transmitted during sessions. GIOP guarantees interoperability between ORB implementations from different vendors.
- GIOP is mapped onto different underlying transport protocols. For example, the Internet Inter-ORB Protocol (IIOP) is a specialised mapping of GIOP to TCP, the Internet transport layer.

Dynamic invocation involves the construction of CORBA requests at run time. The dynamic skeleton interface accepts requests for which it has no skeletons, inspects its contents and invokes the object and method it is targeted for. Hence, applications can make spontaneous interactions of discovered CORBA services.

3.1.3 Web Services

The Web Services Architecture (WSA) [11] is an open standard whose goal is to ensure interoperability between software applications running on a variety of platforms and/or frameworks by utilising the technologies of the World Wide Web. A Web Service is an abstraction, whose service description is documented using the XML-based Web Service

Description Language (WSDL) [10][1]. Each WSDL description is composed of two parts that play an important role in achieving interoperability:

- The *Abstract Section* describes the service independently from protocol specific information in terms of XML data types, message types (four operation types were originally available, however in WSDL 2.0 flexible 'Message Exchange Patterns' allow all types of service interactions to be described) and operation sequences.
- The *Concrete Section* describes how the abstract service is bound to a concrete protocol e.g. how the abstract message are transported using either HTTP or SOAP.

Figure 3.3 illustrates the technologies that underlie the Web Services Architecture. The abstract messages, described in WSDL, are encapsulated into SOAP messages (although the concept of Web Services does not discount other message formats), which may then be bound to different transport protocols (e.g. HTTP, FTP, IIOP, JMS); specifications for SOAP to HTTP and SMTP bindings have been defined. SOAP provides a protocol neutral format for secure, reliable, multi-party messaging; the information needed to invoke remote services can be serialised and transported across the wire and interpreted by the remote service regardless of its platform.

The Web Services architecture is one implementation of Service Oriented Architecture An important role here is service discovery; through open publication, software processes are available to use by a wide audience. Before a service requestor and provider can interact, the correspondents must agree on the service description and semantics of the interaction. Discovery can be performed with or without human intervention; a user can use a suitable discovery tool (c.f. Jini browser), or an autonomous agent can select a suitable service. The Web Services architecture does not specify how the discovery process is to be carried out; it may be a search engine process or a discovery protocol like Jini. However, in practice only a small number of methods have been used e.g. Universal Description, Discovery and Integration (UDDI) [21], a centralised registry for WSDL interfaces.

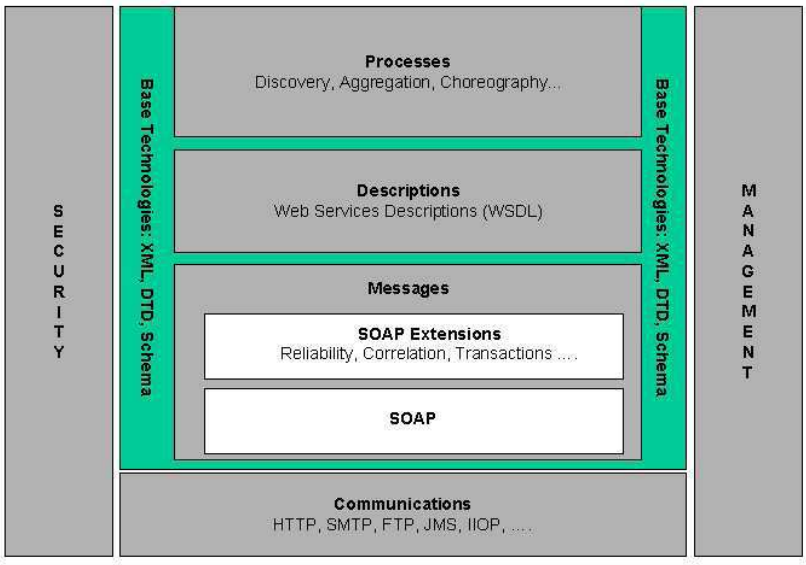


Figure 3.3: Web Services Technologies [11]

[1] WSDL 2.0 is now a recommendation of the W3C and extends upon the earlier version 1.1 (which was not endorsed, but is more prevalent in current software toolkits)

3.1.4 Analysis of Traditional Middleware

Both CORBA and Web Services attempt to make the world conform to a common standard; such an approach has been effective in many areas e.g. routing of network messages in the Internet. And to some extent the two approaches have been successful in connecting systems in Enterprise applications. However, 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:

- 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 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.
- 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).
- 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.

While unsuccessful, it is important to indentify that Web Services and CORBA employ a number of techniques that are important when considering solutions for universal interoperability: i) language independent descriptions of application service behaviour, ii) general protocol message formats, and iii) the separation of abstract services from the concrete middleware implementation is potentially the key to overcoming heterogeneity, and is one of the strong elements of the Web Services approach that has not been fully explored.

With respect to the five dimensions of interoperability, traditional middleware typically achieves the following:

- *Discovery protocol interoperability*. Traditional middleware use a single discovery protocol e.g. the CORBA naming service and UDDI.
- *Interaction protocol interoperability*. Interaction protocol heterogeneity is only resolved where mapping to the common standard is implemented statically.
- *Data interoperability*. Traditional middleware does not resolve data heterogeneity.
- *Application interoperability*. Traditional middleware does not attempt to resolve application-level heterogeneity.
- *Non-functional properties*. Traditional middleware does not attempt to handle interoperability of non-functional properties; they standardise mechanisms for security and transactions (as with interaction protocols).

3.2 Interoperability Platforms

In this section we describe existing interoperability platform solutions. Figure 3.4 illustrates the key elements of approaches that provide an interoperability platform for client, server, or peer applications to be implemented directly upon. For example, for a client side application it guarantees that the application can interoperate with all services irrespective of the middleware technologies they employ, and similarly for server and peer applications. 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.

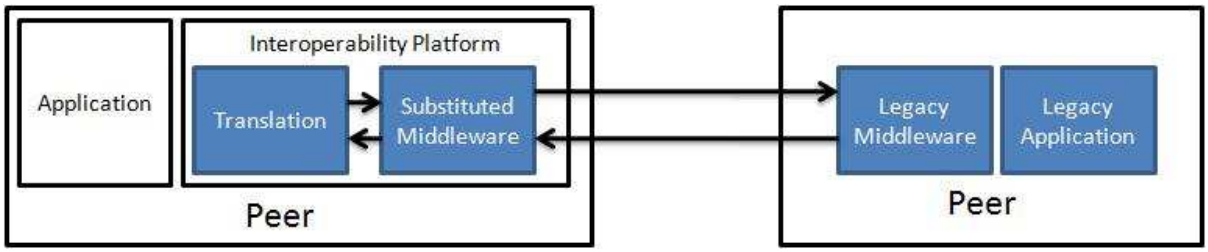


Figure 3.4: Interoperability Platform

This implementation style will vary based upon the end-systems. For examples, clients and peers may realise only a single substituted middleware at a time, whereas a server application may be hosted atop multiples of substituted middleware i.e. so that all types of clients can interoperate with it. 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.

3.2.1 **Universal Interoperable Core (UIC)**

Universally Interoperable Core (UIC) [22] was an early solution to the middleware interoperability problem; in particular it is an adaptive middleware whose goal is 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 is based upon the architectural strategy pattern of the dynamicTAO system [23]; namely, it provides a skeleton of abstract components that form the base architecture. This then enables the platform to be specialised to have the specific properties of particular middleware platforms by adding middleware specific components to it (e.g. a CORBA message marshaller and demarshaller). An example personality is presented in Figure 3.5, which shows a multi-personality platform for communicating with both Java RMI, and CORBA servers.

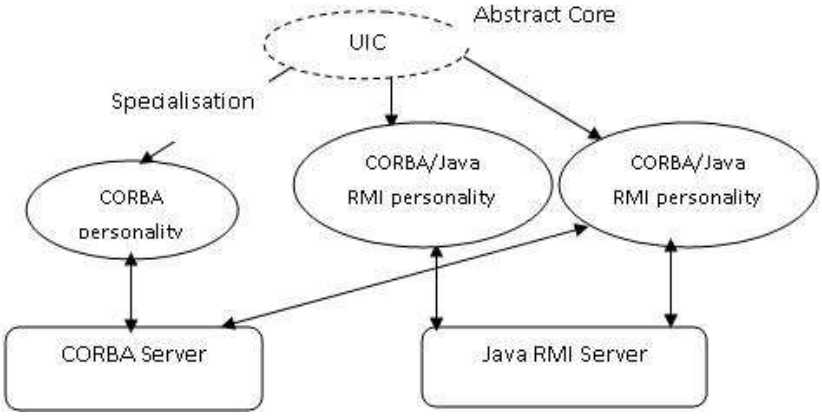


Figure 3.5: UIC Personalities

Analysis of UIC

- *Interoperability dimensions.* UIC has a narrow focus, considering only a single communication abstraction (i.*e. distributed object) and does not* consider other interaction types, discovery protocol or semantic interoperability challenges.

- *Abstraction.* The solution is based upon a common abstraction for distributed object interaction (presented to the developer as an API). A weakness of the solution is that it has only been evaluated for a small number of protocols i.e. CORBA and Java RMI.

3.2.2 Reflective Middleware for Mobile Computing (ReMMoC)

ReMMoC [24] is an adaptive middleware that was developed especially to tackle the problem of achieving interoperability between mobile device applications and the available services in their local environment. In this domain 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 component frameworks as illustrated in Figure 3.6:

- *a service discovery framework* which can be 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.
- *a binding framework* that allows the interaction between services by plugging-in different binding type implementations e.g. an IIOP client, a publisher, a SOAP client, etc.

To determine what protocols are in use, the framework monitors the environment to identify if messages of a protocol are being used (sending dummy messages to trigger this in some cases). If the presence of a service discovery protocol (SDP) is detected, an event is generated and the discovery framework automatically reconfigures itself, loading the corresponding protocol plug-in. When the application makes a service lookup request it is sent across all configured protocols and the results are then used to determine the dynamic reconfiguration of the binding framework e.g. if a SOAP service is found, the binding framework reconfigures to the SOAP plug-in and the application can then interact with the service.

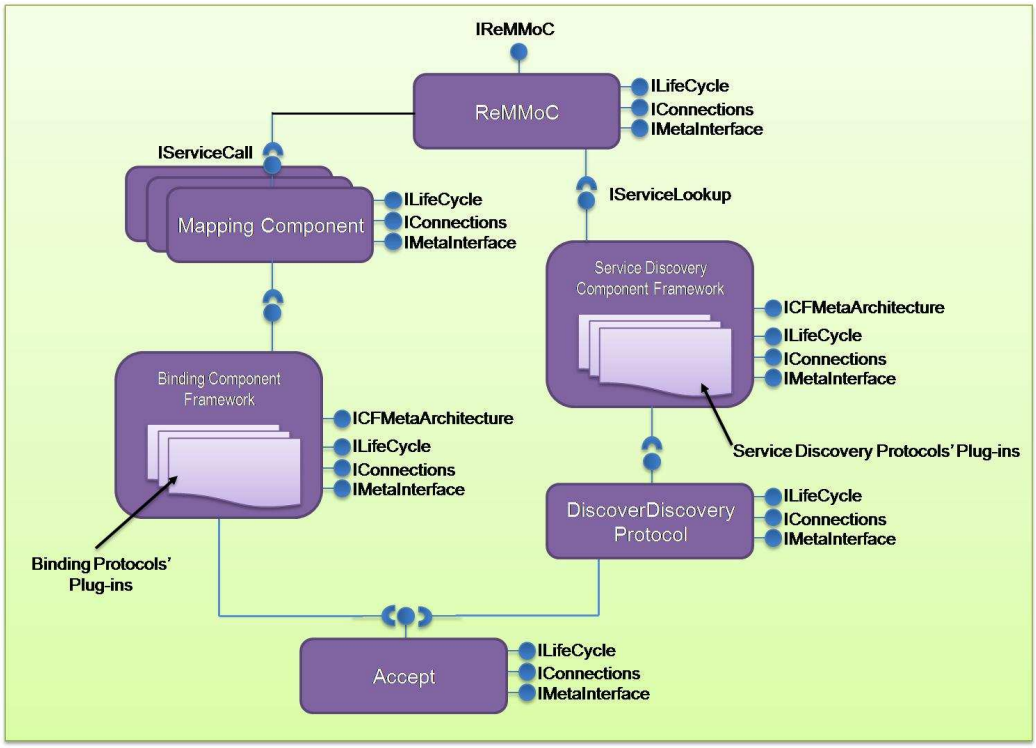


Figure 3.6: The ReMMoC Middleware Architecture.

Analysis of ReMMoC

- *Interoperability dimensions.* The approach considers both discovery and interaction interoperability; in particular it addresses cross-communication abstraction heterogeneity e.g. interoperation between publish-subscribers and distributed objects. The approach however does not consider the semantic and data differences.
- *Abstraction*. ReMMoC employs a service abstraction based around WSDL i.e. the programming model and API provided for the development of client applications ensures interoperation if and only if the service has been implemented to the same WSDL description (at the abstract level at least, any concrete binding information in the WSDL is ignored). Hence it cannot interoperate with services that do not make this assumption.

3.2.3 Web Services Invocation Framework (WSIF)

The Web Service Invocation Framework (WSIF) [25] 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 needs to be extended to understand each format. Hence, the same WSIF client code can, in theory, interact across any available binding. WSIF is a client side framework, none of its implementation resides at the service side, and therefore existing middleware solutions can be used in place. For example, a CORBA service can be exposed as a Web Service by creating and then advertising a WSDL description of the service.

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. Figure 3.7 illustrates the operation of WSIF. A remote service is represented by its WSDL description. The client does not care how this is implemented; it simply needs to obtain the description dynamically, using a discovery process (typically UDDI). The client then loads and parses this to create its representation of the service, which is responsible for generating the abstract operations for the client to invoke. When such an abstract operation is invoked, the WSIF provider takes this information and produces messages through serialisation; these correspond to the described binding mechanism, interact with the remote service and respond with the abstract results.

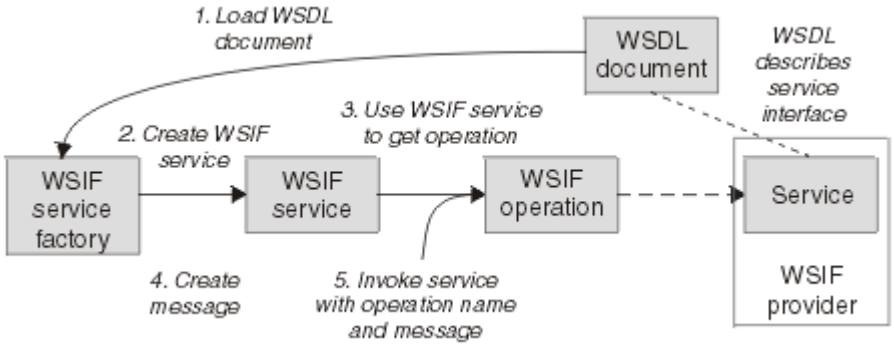


Figure 3.7: The WSIF Client Framework

WSIF provides a remote method invocation programming style; the developer invokes abstract operations and receives their results (although the user can choose for this to be asynchronous). However, this does not take into account the different programming models of

the various underlying middleware it abstracts from. The performance of executeRequestResponse over IIOP, SOAP, EJB and local Java classes will be predictable (a result or fault will be returned), as these follow the RMI paradigm. However, with event based middleware a request may be unanswered for some time, although this does not indicate an error has occurred. Therefore, developing in this style would lead to varied performance of the application depending upon the computational model of the current underlying paradigm.

Analysis of WSIF

- *Interoperability dimensions.* WSIF only considers the interaction heterogeneity problem. Note, WSIF follows the discovery model of Web services, and requires new and existing services to be available through explicit advertising of the WSDL file (e.g. in a UDDI registry). Hence, it is restricted to use with discovery protocols that return a WSDL file. This is different to ReMMoC which has an implicit relationship with WSDL i.e. the file doesn't need it to be advertised and downloaded.
- *Abstraction*. WSIF relies upon service developers sensibly exposing implementation as Web Services (the same philosophy as ReMMoC). The method of wrapping heterogeneous middleware services as Web Services has been criticised because the choreography of individual middleware platforms are not the same as the choreography of Web Services [26]. For example, CORBA is both Service Oriented and Session Oriented. Exposing a session oriented CORBA object would cause specific CORBA implementation details, like remote object references, to appear in the WSDL abstraction.

3.2.4 Analysis of Interoperability Platforms

For the particular use case, where you want a client application to interoperate with everyone else, interoperability platforms are a powerful approach that has demonstrated this is achievable. Within this scope the interoperability challenges have been resolved as follows:

- *Discovery protocol interoperability.* ReMMoC has shown that a client can discover services irrespective of the service discovery protocol used to advertise.
- *Interaction protocol interoperability.* ReMMoC and WSIF have demonstrated that clients can interact with a range of communication abstractions from RPC to Publish-Subscribe.
- *Data Interoperability.* None of these approaches has explicitly considered data heterogeneity.
- *Application interoperability*. No of these approaches has explicitly considered application-level heterogeneity.
- *Non-functional properties*. None of these approaches has explicitly considered non-functional properties.

Further, 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.

3.3 Bridging

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. Figure 3.8 illustrates these key elements. Note, appropriate discovery and naming services are used to ensure that an application is pointed to the bridge (rather than the original service endpoint); hence additional middleware deployment is required to complete the interoperability solution (the bridge alone is insufficient).

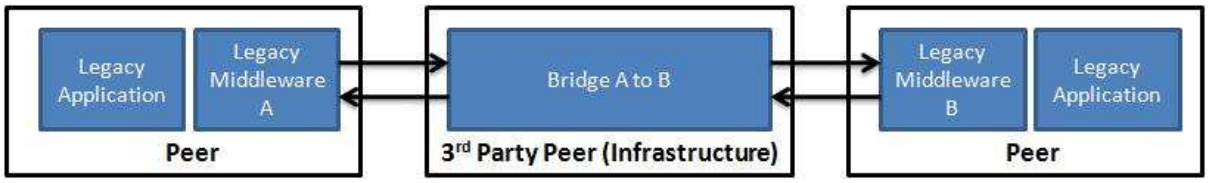


Figure 3.8: Software Bridges

Many bridging solutions have been produced between established commercial platforms The OMG has created the DCOM/CORBA Inter-working specification [27] that defines the bi-directional mapping between DCOM and CORBA and the locations of the bridge in the process. OrbixCOMet [28] is an implementation of the DCOM-CORBA bridge. SOAP2CORBA [29] 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 them.

3.3.1 Model Driven Architecture

Model Driven Architecture (MDA) [30] is an OMG specification that aims to support interoperability and integration throughout the systems lifecycle. MDA defines how to specify a system in terms of system functionality, separated from its implementation on a particular platform. To perform this, MDA is separated into key models, which are shown in Figure 3..9. For creating MDA-based applications, the first step is to create a Platform Independent Model (PIM), which is expressed in UML. The PIM provides a formal specification of both the structure and function of the system, which is abstract from any technical details. Similarly, the Platform Specific Model (PSM) defines in UML how a PIM is realised on a particular platform e.g. EJB/CORBA, as shown in Figure 3.9. This mapping of PIM to PSM UML descriptions can be automated for standard mappings (each platform specific model is then physically implemented). Finally, the integration between alternative PSM implementations can be overcome by the automated insertion of a suitable pre-developed bridging solution.

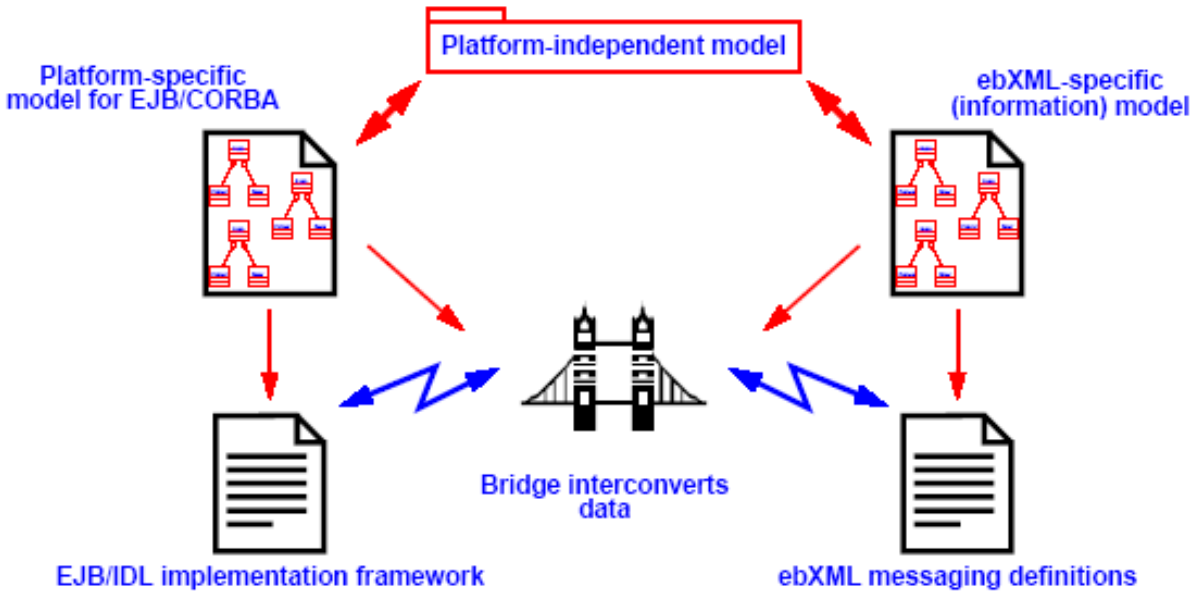


Figure 3.9: OMG's Model Driven Archtecture [30]

MDA is a powerful tool for specifying systems that may be composed of heterogeneous elements. The complete architecture can be designed at the abstract level and this viewpoint

does not consider heterogeneity problems. Rather the automation of platform specific implementation (e.g. integrated through bridges) carries this out. As with Web services, the solution to overcome heterogeneity is to provide a higher-level abstraction. However, the model is suited to system design and initial configuration; it does not deal with unforeseen changes in heterogeneity during the lifecycle.

Analysis of MDA

- *Interoperability dimensions.* MDA has generally focused on resolving the problems of interaction interoperability. However, the general nature of a platform independent to platform specific model has the potential to be applied to all dimensions of interoperability.
- *Abstraction***.** Bridging offers a solution to connect heterogeneous middleware, however it is a low level mechanism that must be supported by a higher-level abstraction (cf. The platform independent model) to fully support the integration of multiple platform types. However, such a model is reliant on all parties contributing at design time; this leaves many of the challenges of interoperability for spontaneous interactions unresolved, and is more suited to connection of fixed legacy systems in the Enterprise.

3.3.2 Unified Component Meta Model Framework (UNIFrame)

The UNIFrame approach [31] attempts to unify distributed component models under a common meta-model to allow discovery, interoperability and collaboration between components using generative programming techniques. The key parts of the framework are: the Unified Meta Model (UMM), the Unified Component Interoperability framework (UCI) and automated system generation. The UCI framework is the technology involved in overcoming heterogeneity, so is described in further detail.

The UCI allows for the static and dynamic assembly of heterogeneous components. The architecture, described in Figure 3.10, consists of platform independent formal specifications and a heterogeneous component integrator. The formal specification contains both the functionality and QoS contracts of the component. The component integrator is made up of a translator, an internal representation and the Middleware Bridge Generation Engine (MBGE). The translator takes the platform specific component specification and creates a platform independent specification. Then as seen in Figure 3.10, the abstract representations of two components can be supplied to the MBGE to automatically produce a bridge between them to allow them to interoperate.

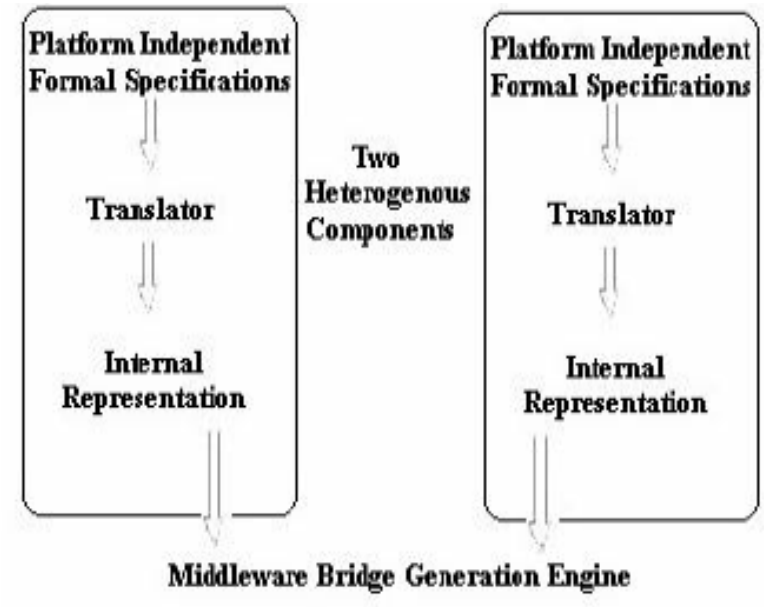


Figure 3.10: Architecture of the Unified Component Interoperability framework

UNIFrame is similar to MDA. However it differs in that the independent model is generated from the specific model (rather than the other way round). Furthermore, the architecture allows for the dynamic creation and insertion of bridges to overcome heterogeneity.

Analysis of UNIFrame

- *Interoperability dimensions.* UNIFrame considers already discovered components and hence focuses on achieving interoperability in creating the interaction connectors between them.
- *Abstraction*. A platform independent model of components and connectors is again used; however, this time the model is generated at runtime from the already defined platform specific models i.e. these must still be created at design time. The models then input to the dynamic generation of bridges. Hence, it overcomes the problems of creating static bridges.

3.3.3 Enterprise Service Buses (ESB)

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 [32], BEA Aqualogic[2] and IBM Websphere Message Broker[3].

Analysis of ESBs

- *Interoperability dimensions.* ESBs concentrate solely on the individual messaging communication abstraction. To perform discovery UDDI is generally utilised; clients search UDDI servers for services that match their functional service requirement and then they connect to them via the message bus.
- *Abstraction***.** The assumption is that all messaging services can be mapped to the ESB intermediary messaging abstraction (which is a general subset of messaging protocols). This decision is enacted at design or deployment time, as the endpoint must deploy code to connect to a particular message bus with an appropriate translator.

3.3.4 MUSDAC

The MUlti-protocol Service Discovery and Access (MUSDAC) middleware platform [33] enables clients to interact with services built on top of heterogeneous discovery and access protocols hosted on different IP networks that do not necessarily have a direct connection. The functionality of this middleware is provided as a service which can be accessed via any existing SDP to access MUSDAC, the service is first registered in all active discovery domains within the local network. Then, it can be accessed using any of the available discovery and access protocols (e.g. SSDP and SOAP for UPnP). Hence, it is a middleware to deploy clients upon (just like an interoperability platform), but also relies on services in the infrastructure to handle the interoperability (hence, we classify it as a bridging solution).

Figure 3.11 shows an overview of the MUSDAC architecture. The MUSDAC platform deployed in the infrastructure is composed of four components: Managers, Service Discovery and

[2] http://dev2dev.bea.com/aqualogic

[3] www.ibm.com/websphere/wbimessagebroker

Access (SDA) Plugins, Transformers and Bridges. Managers handle local and remote discovery and access requests. The SDA Plug-in interacts with specific SDPs collecting service information, registering the MUSDAC service and performing service access on behalf of remote clients. Transformers extend the service description generated by SDA Plug-ins with context information. Finally Bridges enable communication with other networks in the greater pervasive environment. Depending on each device's capabilities, they register to potentially support some or all of the instance components.

To locate a service, clients first discover the MUSDAC service using the SDP of their preference. Then, a MUSDAC request is sent to the discovered MUSDAC instance using the client API provided. The MUSDAC service that receives the request forwards it to the local manager which, in parallel, communicates the request to all available discovery domains using the local SDA plug-ins for locating services in the local network and to the local bridges to propagate the request to nearby networks. A manager receiving a remote service request (i.e. from a bridge) processes it as a local one and returns the response to the originating bridge. Finally, the requesting manager collects local and remote responses and sends them back to the requestor.

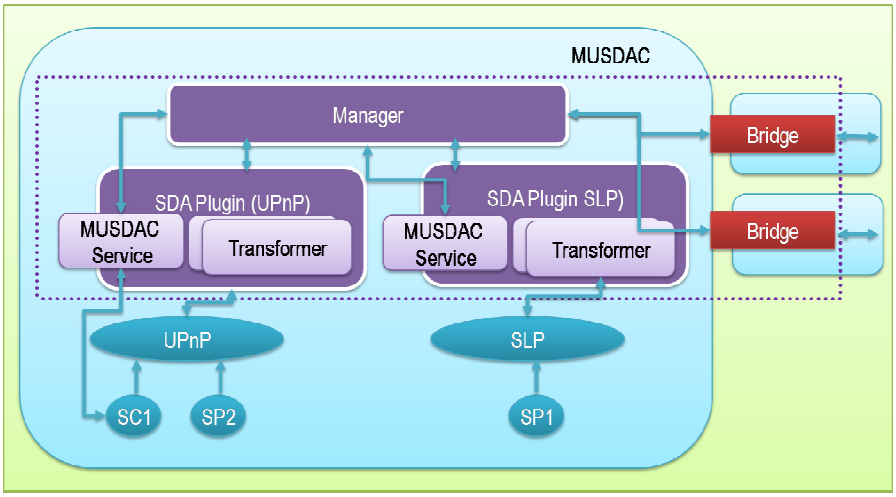


Figure 3.11: Overview of the MUSDAC platform

Analysis of MUSDAC

- *Interoperability dimensions*. MUSDAC resolves only service discovery protocol interoperability issues.
- *Abstraction*. MUSDAC defines a generic and modular service description format as the basis for mapping between protocols. A MUSDAC service description contains: i) creation information that documents the components that generated the description and where it was generated (i.e., IDs of Manager, SDA plugin and Transformers); ii) service information containing the service properties as specified by the original SDP (e.g., service name, methods names and input/output parameters if available); iii) service context information indicating the context properties and admission control policies associated with the service instance; and iv) propagation information specifying the route path in terms of networks and bridges between the provider and consumer.

3.3.5 Analysis of Bridging

Bridging solutions have shown techniques whereby two protocols (discovery or interaction) 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; however, the technologies themselves are limited in the dimensions they consider.

- *Discovery protocol interoperability*. MUSDAC is the only solution that resolves discovery protocol interoperability.
- *Interaction protocol interoperability*. Bridging is a common solution to interaction protocol interoperability e.g. as shown by ESBs, MDA and UNIFrame.
- *Data interoperability*. No bridging solution described investigates the problem of heterogeneous data.
- *Application interoperability*. No of these approaches has explicitly considered application-level heterogeneity.
- *Non-functional properties*. None of these approaches has explicitly considered non-functional properties.

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.

3.4 Transparent Interoperability

In the interoperability platform approach at least one endpoint is aware of the interoperability problem and employs a framework to resolve it. In transparent interoperability solutions neither application is aware, and hence legacy applications can be made to communicate with one another. Figure 3.12 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').

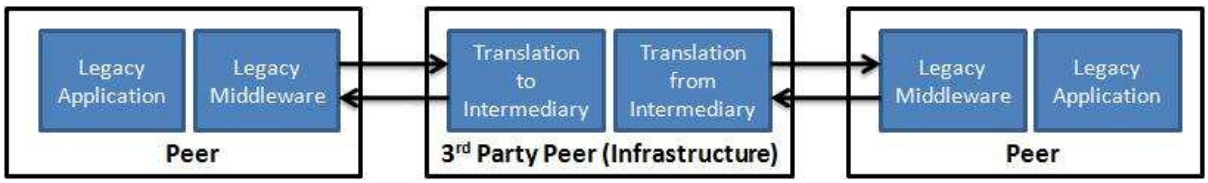


Figure 3.12: Transparent Interoperability

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 that are transparent to the application); however, they are commonly deployed across one or more infrastructure servers.

3.4.1 INDISS/Nemesys

The INteroperable DIscovery System for networked Services (INDISS) system [34] is a service discovery middleware based on event-based parsing techniques to provide service discovery interoperability in home networked environments. INDISS is deployed as part of a service provider, consumer, registry, or independently in some intermediate networked node; importantly it is transparent to applications which are implemented upon legacy discovery protocols. Hence, INDISS acts as glue between protocols.

First, 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. Events are defined based on conceptual similarities among SDPs. Because communication between parsers and composers does not depend on any syntactic detail of any protocol, a parser for a given protocol can communicate with composers from any other protocol and vice versa. Figure 3.13 illustrates an overview of INDISS' SDP detection and interoperability mechanisms.

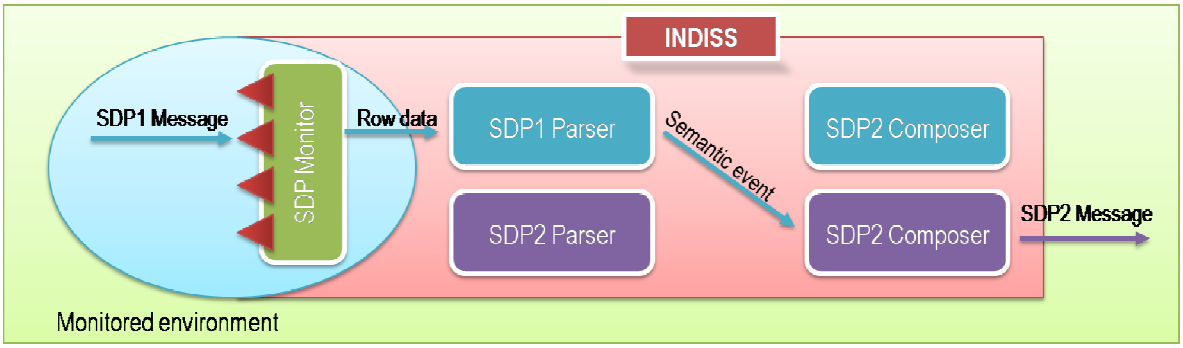


Figure 3.13 INDISS' SDP detection and interoperability mechanisms

INDISS functions as a bridge deployed in the infrastructure connecting different SDPs without setting up dependencies with client applications. This is why INDISS instances can be deployed independently in the environment and applications are not required to use any API; and interoperability of legacy applications can be handled.

Analysis of INDISS

- *Interoperability dimensions.* The approach is best suited to service discovery protocols. The Nemesys extension of INDISS utilises similar techniques to transparently bridge between different RPC protocols; however, it has only been evaluated for two protocols (SOAP and Java RMI) with a reduced type system (3 primitive types)—hence its generality needs to be proven.
- *Abstraction*. INDISS employs a common abstraction of discovery protocols, i.e., the common event intermediary that each protocol is mapped to; this is considered the minimum subset of all protocols (as opposed to substitution and direct 1-1 mappings whereby more of the protocol functionality is considered). Further, the abstraction exists at the interoperability level and is hidden from the applications.

3.4.2 uMiddle

uMiddle [35] 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. Multiple instances of uMiddle may be deployed on the same networked environment. Services connected to these instances, or discovered by them, can be freely and transparently utilized by services or by consumers known to other uMiddle instances. This functionality is supported by the directory module

which is responsible for managing the exchange of service advertisements between discovery agents providing discovery mechanisms that support the notification of other uMiddle instances about the presence of services. Notifications are independent of the actual discovery protocols utilized. A transport module is also provided to enable communication among translators.

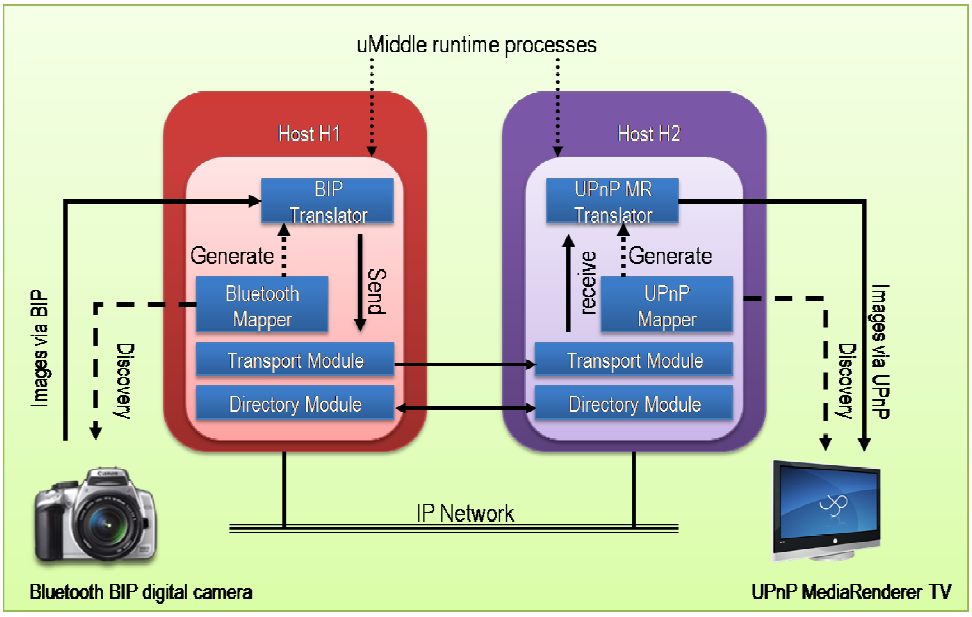


Figure 3.14: uMiddle's system architecture

Figure 3.14 depicts the uMiddle system architecture for translating between Bluetooth and UPnP. Services are described using a XML-based language called the Universal Service Description Language (USDL). USDL serves as means for the runtime configuration of a translator based on the required service semantics. However one translator implementation for each device is still required. The uMiddle design takes into consideration the architecture of SDPs such as UPnP, Bluetooth and Jini. The uMiddle platform is similar to INDISS in that client applications do not need any modification or using a specific API, it has to be deployed in the infrastructure and it functions independent of applications (i.e. applications do not need to be aware of its prescence).

Analysis of uMiddle

- *Interoperability dimensions.* The approach considers both discovery and interaction interoperability; however, only discovery is presented in detail and there is limited investigation of applying the solution to a range of binding protocols.
- *Abstraction*. A universal language is used to describe all services; this is used as the basis for translating to and from service discovery protocols

3.4.3 OSDA

The Open Service Discovery Architecture (OSDA) [36] 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. As Figure 3.15 shows, the OSDA system assumes that discovery agents (i.e. the service registry, service consumer and service provider) are already in place.

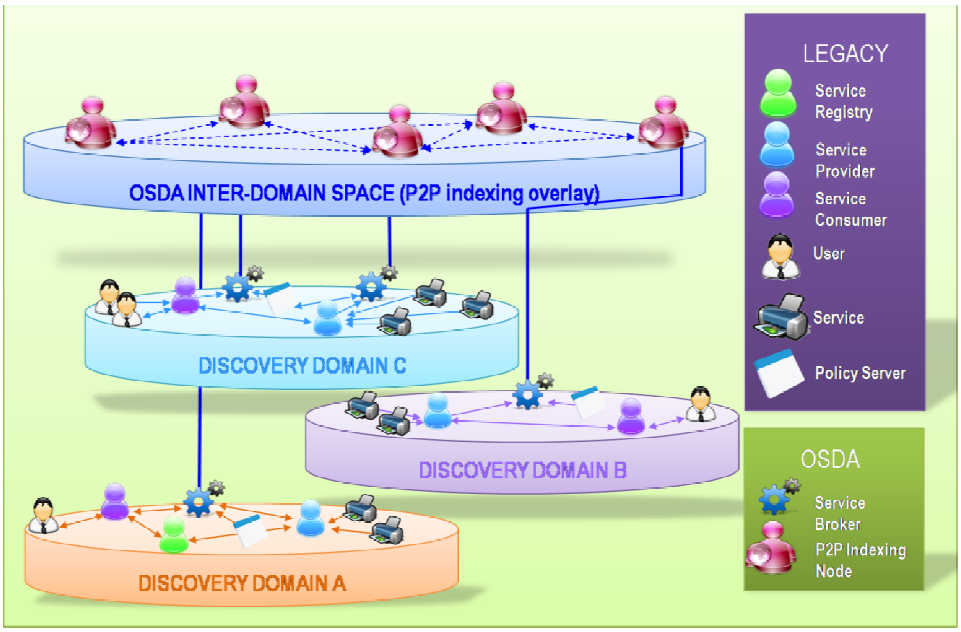


Figure 3.15: High level overview of OSDA

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. Service brokers are divided into two layers: i) a technology dependent layer which abstracts the local discovery system, intercepting and processing registrations and requests and translating them using the OSDA unified description scheme, and ii) a technology independent layer which is responsible for handling broker-to-peer and broker-to-broker communication and providing the interfaces for the broker to be accessed by the entities involved in the OSDA inter-domain discovery process. The peer to peer indexing overlay provides the inter-domain and inter-network space where services from heterogeneous domains can be registered and queries solved utilizing a Distributed Hash Table.

To register services in the OSDA system, one or more brokers are deployed on each discovery domain. Brokers intercept local service registrations and translate them into a common, well-defined service description format. Then the registration is sent to the peer to peer overlay which distributes the service information. In addition, to locate a service the service request is first translated to a common format and forwarded by the broker to the peer to peer overlay. The broker receives back the set of broker URLs that have been registered along with the services matching the request. Then, the same broker contacts one or more brokers from the received list to retrieve the complete information about the requested service.

Analysis of OSDA

- *Interoperability dimensions.* OSDA focuses solely on functional discovery protocol behaviour.
- *Abstraction*. To enable interoperability among heterogeneous service discovery systems, OSDA proposes the Unified Service Description (USD) scheme. USD is an XML-based scheme which, according to their designers, can be mapped to any service description scheme including WSDL.

3.4.4 SeDiM

SeDiM [37] 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 (as illustrated in Figure 3.16 where a bridge in the infrastructure and a bridge on a local device combine to translate between protocols).

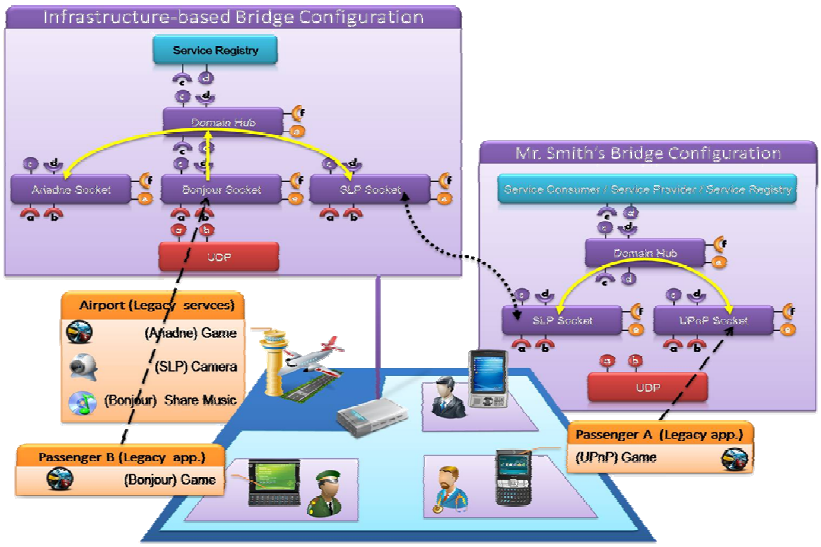


Figure 3.16 SeDiM deployed to translate between 5 protocols.

SeDiM's component framework is similar to the UIC philosophy it provides a skeleton abstraction for implementing discovery protocols which can then be specialised with concrete middleware components to create complete implementations of the protocols. These configurations can then be 'substituted' in an interoperability platform or utilised as one side of a bridge. To then perform the translation two key elements are embraced:

1. Domain Sockets. These are special components that map protocol specific messages to an intermediary format understood by SeDiM (the discovery event model) and vice versa.
2. Domain Hubs are components that map discovery events onto other protocols to achieve interoperability e.g. when it receives an event from SLP it forwards to UPnP.

Analysis of SeDiM

- *Interoperability dimensions.* SeDiM focuses on both functional and non-functional discovery protocol behaviour.
- *Abstraction*. SeDiM has two key abstractions to underpin interoperability. First, a service description language for mapping between description formats to ensure that discovery requests can be matched by different protocols; second, the event abstraction that matches the functional discovery events/operations of heterogeneous protocols.

3.4.5 Analysis of Transparent Interoperability

The solutions in this section illustrate how interoperability can be achieved between two legacy-based platforms; and in this sense they meet the requirements for spontaneous interoperability. However, the technologies are limited in the dimensions of interoperability they consider:

- *Discovery protocol interoperability*. The solutions largely concentrate on solving the discovery protocol heterogeneity problem.

- *Interaction protocol interoperability*. No solution fully explores transparent interoperability of interaction protocols (beyond a few simple cases) and do not consider interoperability across abstractions.
- *Data interoperability*. Semantic and data interoperability has not been investigated in these solutions.
- *Application interoperability*. No of these approaches has explicitly considered application-level heterogeneity.
- *Non-functional properties*. Only SeDiM has considered non-functional properties; however this is only for heterogeneous privacy requirements in discovery protocols.

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 Figure 3.17.

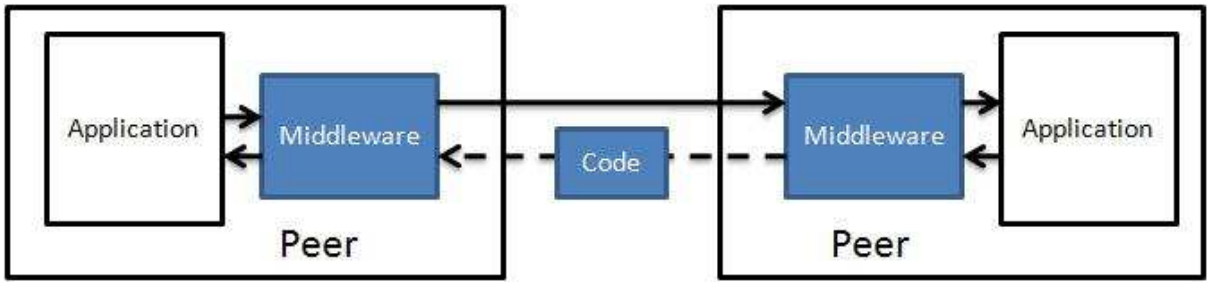


Figure 3.17: Logical mobility

3.5.1 SATIN

SATIN [38] is a low footprint component based middleware, which aims to address the problem of heterogeneous service implementations in dynamically changing mobile environments. It argues that the use of logical mobility (code mobility) is limited within current mobile middleware platforms, but offers genuine benefits for interoperability.

In a scenario where a mobile host is able to access the local services of an ad-hoc network, the peer should be able to obtain code to discover its required services using the discovery mechanism in place and then use it. To do this, the SATIN architecture composes applications and the middleware itself into a set of capabilities (a unit of functionality), for example, a discovery mechanism or compression algorithm. Capabilities are registered with the host's core, which can be statically or dynamically configured. At the heart of SATIN is the ability to advertise and discover service implementations that may be advertised using different techniques; each discovery mechanism is represented by a different capability that can be added to the host when needed in the environment. SATIN then utilises its own "higher level" XML based discovery mechanism for initialisation; that is, the advertising mechanisms currently in use can be discovered. For example, a host uses SATIN to find the discovery capabilities being used and then downloads these. The required application services are looked up and their interaction capabilities are downloaded to complete the cycle.

Analysis of SATIN

- *Interoperability dimensions.* Using mobile code ensures that all interoperability dimensions can be covered in principle. However, the approach focuses on interaction and makes the assumption that a common discovery protocol is used (the Satin platform).
- *Abstraction*. The underlying abstraction is the use of a common component platform that can exchange code between the endpoints. Hence, in that respect the approach is similar to traditional middleware platforms hosted at each end.

3.5.2 Jini

Jini [39] 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.

The Jini architecture centres on central federated lookup services that physically exist on remote machines in the network domain; clients and services first discover lookup services in their vicinity before utilising them. The lookup service consists of a directory of service items, which are made up of three elements: 1) its service interface (defined as a Java Interface), 2) a Java object (service proxy) on which calls to use the service can be made, and 3) a set of service attributes that describe the service. In order to be discovered, new services register this information to one or more lookup services. Furthermore, Jini employs the concept of leasing; a service registers itself for a given time period, called a *lease*. When the lease expires the service is no longer advertised.

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:

- The proxy object may encapsulate the entire service. This strategy is useful for software services requiring no external resources.
- The downloaded object is a Java RMI stub, for invoking methods on the remote service.
- 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; one implementation of a service might be RMI-based, and another CORBA-based. This offers one particular solution to the problem of middleware heterogeneity through the use of mobile code to manage interactions.

Analysis of Jini

- *Interoperability dimensions.* Jini is a single discovery protocol (others are not considered) and interaction via downloaded code is provided only. Hence, all parties must use Jini in order to interoperate.
- *Abstraction*. The underlying abstraction is the use of a common dynamic proxy platform that can exchange code between the endpoints.

3.5.3 Analysis of Logical Mobility

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. In terms of addressing each of the three dimensions:

- *Discovery protocol interoperability*. Logical mobility cannot resolve discovery interoperability in full because it relies on a common discovery platform as the basis of two parties finding each other before exchanging code.
- *Interaction protocol interoperability*. As the interaction protocol can be downloaded onto the corresponding device, the approach has the potential to fully resolve interaction heterogeneity. However, it cannot resolve interaction problems between legacy applications that do not employ code mobility.
- *Data interoperability*. The approach relies on the fact that the application will understand the downloaded code, and does not address the potential for there to be semantic mismatches between the two applications and the mobile code shared.
- *Application interoperability*. No of these approaches has explicitly considered application-level heterogeneity.
- *Non-functional properties*. No approach has considered non-functional properties.

Overall, logical mobility is the weakest of the interoperability approaches due to the reliance on all application conforming to the common middleware. SATIN offers an improved solution over Jini in that the problem of heterogeneous discovery mechanisms is considered yet it still relies on a base, non-standardised, discovery abstraction.

3.6 Semantics-based Interoperability

The interoperation solutions proposed above concentrate on the middleware level. They support interoperation by abstract protocols and language specifications. But, by and large they ignore the data dimension. As highlighted in Section 2.2.4, for two parties to interoperate it is not enough to guarantee that the data flow 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 [40] 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 DB community where there is often the need to access information on different DBs 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.

These efforts are briefly reviewed below, where we look at the Semantic Web services efforts first, then at their application to middleware solutions, and finally at the database experience.

3.6.1 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 labelled as *Semantic Web Services*, attempted 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.

3.6.1.1 OWL-S

DAML-S [41] and its successor OWL-S [42] [43] have been the first efforts to exploit Semantic Web ontologies to enrich descriptions of services, with the intent. The scope of OWL-S was 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.

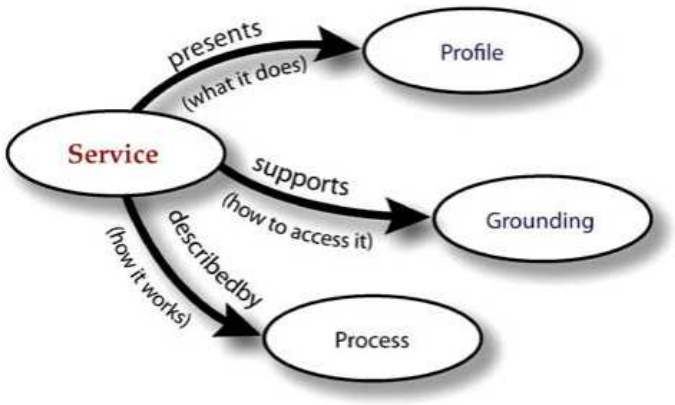


Figure 3.18: OWL-S Upper Level Structure

As shown in Figure 3.18, 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 *input* 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 distinguished in two categories: atomic processes that describe atomic actions that the service can perform, and composite processes that describe the workflow control structure.

In turn atomic processes “ground” into WSDL operations as shown in Figure 3.19 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, therefore, at least in principle, every service which can be expressed in WSDL, can also 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. Therefore, it is challenging to address the problem of mediating between the `GetInfo()` operation of a service and the `GetLocation()`, `GetPrice()`, and `GetQuantity()`of another service. 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 between services. 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? OWL-S is mute with respect to this question, but ontology merging techniques described in Section 5.4 can be exploited to derive such an ontology.

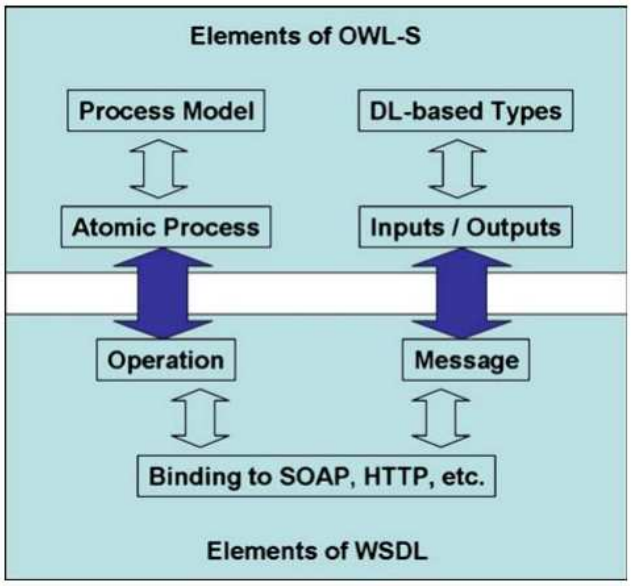


Figure 3.19: The structure of the OWL-S process grounding

3.6.1.2 SA-WSDL

Semantic Web services reached the standardization level with SA-WSDL [44], 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 [45] 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 three new attributes:

- *modelReference*, to specify the association between a WSDL or XML Schema component and a concept in some semantic model.
- *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 [46] 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 [47].

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 result that at 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 give the message of the service to derive its semantic representation.

3.6.1.3 WSMO

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) [48], which provides the formal language for service representation, and WSMX (Web Service Modelling eXecution environment) [49] 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 Web service representation framework concentrated on the support of the different operations that can be done with Web services, namely discovery with the Service Profile as well as UDDI [21], composition with the Process Model as well as BPEL4WS [50] and WS-CDL [51], and invocation with the Service Grounding, WSDL or SA-WSDL, WSMO provides a general representation of services that can be utilized to support the operations listed above. To this extent it identifies four core elements shown in Figure 3.20:

- **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.

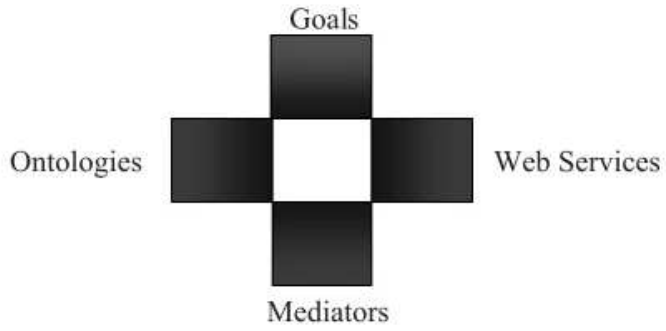


Figure 3.20: WSMO Core Elements

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.

These aspects are of particular interest for CONNECT, especially the definition of mediators, since they bear strong relations with the *CONNECTors* that are the expected result of the project. WSMO defines 3 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.

Starting from this initial classification, WSMO defines 4 basic types of mediators between its foundational elements. Specifically, it defines:

- **OO Mediators** resolve mismatches between ontologies and provide mediated domain knowledge specifications to the target component.
- **GG Mediators** connect goals, allowing the creation of a new goal from existing goals and thus defining goal ontologies.
- **WG Mediators** link a Web service to a goal, resolves terminological mismatches, and states the functional difference (if any) between both.
- **WW Mediators** are used to establish interoperability between Web services that are not interoperable a priori.

From the point of view of the infrastructure that is emerging for CONNECT, OO-mediators address the problem of data interoperability, while WW-Mediators address the problem of mediators synthesis, at least at the service/component protocol level. The WG mediator specifies additional conditions that need to be expressed to allow a client to work with the service, while it is unclear whether in CONNECT there will be any need for an explicit notion of goal and of GG-mediators.

Analysis of WSMO

WSMO put a strong emphasis on mediation and, as discussed above, it defines mediation as "first class" citizen. The problem with WSMO is that that the WSMO project proposed an execution semantics for mediators [52] [53] [49] but so far no theory or algorithm on how to construct mediators automatically has been proposed by the project. Somehow, it is somewhat 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.

Given this analysis, it is clear that the contribution that CONNECT will make will be above and beyond the WSMO efforts. The objective of CONNECT is to build CONNECTors at run time. Therefore, with respect to CONNECT, WSMO may provide a way to represent mediators, but it does not contribute a way to construct them.

3.6.1.4 Exploiting Semantic Web Services

Semantic Web services have been used to address a number of problems. The first problem has been the discovery of services on the basis of their semantic description. Second, they have been used to derive service compositions that exploit very different services; third, they have been used for verification of compositions.

The original motivation for semantic in conjunction with Web services has been to support automatic service discovery and composition [54] [55]. The goal of this work is to provide a form of composition that is essentially client driven. Specifically, the client starts with a formal specification of a goal that it wants to achieve, and builds the service composition to achieve this goal. The construction of such a composition is typically defined as a back-chaining process inspired by the planning algorithms defined in Artificial Intelligence.

Indeed, the relation between the composition process and planning process is close. Many of the most successful composition mechanisms have been built on planning processes. Among them [56] is based on the SHOP2 HTN planner that progressively decomposes an initial task into smaller tasks until it achieves the tasks that can be performed through service operations. An alternative that has also been quite successful has been to rely on the Golog planner which is based on the situation calculus [57] [58] as well as state-of-the-art planners [59].

During composition, the semantic service representation, and the underlying logic are used in two ways: first, to verify that all information that is required to fill the inputs of all operations will be available at the time of execution, and second, that all operations' preconditions will be satisfied at the time of execution. When all inputs are available and preconditions are satisfied the planning process is stopped and the composition is executed. Otherwise the planning process continues by looking for services that can satisfy the given precondition or provide the expected input. In the latter case, the planner would trigger a discovery process to find the services whose outputs and/or post-conditions fit the open inputs and preconditions. Since two services may represent input and output data, as well as preconditions and effects, in very different ways semantics helps in abstracting the differences and in discoverying services that provide information that is "close enough" to what the composition process requires.

The evaluation of what accounts for "close enough" is one of the main problems of the semantic discovery process. The CMU matchmaker [60] [61] provided an initial answer by defining a number of different levels of match depending on the logical relation between the information requested by the client and outputs of the services in the system. Since that initial work, a number of services have been proposed [62] [63] improving on the original techniques to achieve better precision and recall.

Service composition, when defined as the result of a planning process, is always client driven in the sense that at each step of the composition the client interacts with a service and then eventually uses the information received to interact with other services. Crucially, in the process there is never a direct interaction between two services within the same composition as currently analyzed by WP3 of CONNECT.

3.6.2 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 [64] 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) [65] 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 Ebiquity group describes a semantic service discovery and composition protocol for pervasive computing. The service discovery protocol, called GSD (Group-based Service Discovery) [66], 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 [67]. 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 [68] and [69] 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.

3.6.3 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*" [70]. Federated Data sources have a lot in common with the heterogeneous systems to be connected within the CONNECT system. 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) [70] 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. This is a quite expensive proposition and definitely not applicable to the run-time interoperability that is the goal of CONNECT.

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 [71] [72]. 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 [73], 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 problem of these approaches within CONNECT is that it is quite difficult to understand how inaccuracy can be propagated across CONNECTors. For example, schema mapping tools may be able to identify that an input `price` of a component may be equivalent to the output `cost` of another component with some level of accuracy K. One challenge then is to figure out how accuracy K will affect the accuracy of the whole CONNECTor, and of the connected system as a whole.

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 [74] 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 [75].

3.6.4 Analysis of the Semantics-based Interoperability Approaches

Above we reviewed a wide range of semantic and data interoperability approaches that range from approaches that have been motivated by the data interoperability within the DB federation field, to the data interoperability approaches in the Web services field; furthermore, approaches to semantic middleware attempt to apply semantic techniques to other service-oriented middleware platforms. The result of this analysis is a number of data interoperability

solutions that range from hard-coding transformation rules, to learning these rules, to exploiting ontologies to derive mapping rules.

The closest to provide an initial solution to the problem of data interoperability comes from the Semantic Web services field, where WS languages have been generalized to represent the semantics of the input and output messages of the service operations. Ideally, the semantics of input and output messages is then analyzed during the composition process and integrated.

Whereas this is the theory, in practice the composition processes of semantic Web services are still very primitive and since they concentrate on the planning part, essentially they are more about control flow than data interoperation. Indeed, usually the reasoning components that are required for data interoperations, such as the OWL inference engines, are not even present in the composition process. Similarly, and somewhat surprisingly, the few efforts on verification [76] [77] of service representation and compositions concentrate on the process level verification completely ignoring the verification of the data level.

In addition to the technical problem of combining data interoperation with processing interoperation, there are a number of fundamental problems when using ontologies to process data. The first, and most obvious, is that ontologies move the interoperability problem "one level up" in the sense that even if we were able to do data interoperability using ontology, we would still have a problem of ontology interoperability. Essentially: what guarantee of interoperability is there if two systems use different ontologies? We touched upon this problem in reference to OWL-S, and noticed that WSMO introduces mediation across ontologies. In Section 5.4, we describe some experiments in which we try to analyze the problem of ontology interoperability, but any solution to this problem, if possible at all, is far from being reachable. On the more practical side, ontologies come with a high level of computational complexity which in the worse case may be high in the exponential. This raises problem with respect to the practical applicability of such high complexity technology. In addition there is a problem of who constructs ontologies, and what guarantees are there that they will be sufficiently extensive to represent all the information required to process the systems data. To address the latter problem there are a number of efforts that go under the label of "linking open data" [78] which collects billions of ontological statements, but their relation with services and middleware is still to be addressed.

Ultimately the problem of data interoperability is still open, but at least Semantic Web services provide a blue-print on how to map the data and control sides of the equation.

Overall, with respect to the five interoperability dimensions, the semantics-based interoperability approaches address the following:

- *Discovery protocol interoperability*. The solutions generally require a single discovery mechanism e.g. UDDI for discovering WSDL files and do not consider heterogeneity here.
- *Interaction protocol interoperability*. The use of multiple concrete protocols from the abstract service descriptions of WSDL (c.f. Web Services) mean that the approaches can perform some interoperability between heterogeneous protocols.
- *Data interoperability*. The approaches cover a number of semantic-based mechanisms.
- *Application interoperability*. The approaches cover a number of semantic-based mechanisms.
- *Non-functional properties*. No of these approaches has explicitly considered non-functional properties.

3.7 Summary

The results of this state of the art investigation 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 scenarios 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.

With respect to the second problem, namely addressing the interoperability requirements of the scenarios 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 3.1 (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, as discussed in 3.7.1 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.

Further, the abstraction column demonstrates that (at some level) conformance to a particular abstraction is required to achieve interoperability. That is, IDL and WSDL are the abstractions that applications are developed with; or different middleware are mapped to a common abstraction independent of the application e.g. the service discovery models and descriptions for the transparent interoperability solutions. These themselves are typically focused to a particular abstraction type e.g. service-orientation; hence no solution covers the full diversity of communication abstractions e.g. making tuple spaces, message-based, RPC and publish-subscribe systems interoperate in a spontaneous transparent fashion.

To conclude, these results show that there is significant potential for the CONNECT project to extend beyond the state of the art in interoperability middleware.

Table 3.1: Evaluation summary of effectiveness against each interoperability dimension

SD = Discovery
I = Interaction
D= Data
A = Application
N=Non-functional

	SD	I	D	A	N	Abstraction	Transparency
CORBA		X				IDL Interfaces	CORBA for all
Web Services		X				WSDL Services	WSDL & SOAP for all
ReMMoC	X	X				WSDL Services	Client-side middleware
UIC		X				RPC	Client-side middleware
WSIF		X				WSDL Services	Client-side middleware
MDA		X				Soft. Arch.	Platform Independent models
UniFrame		X				Soft. Arch.	Platform Specific models
ESB		X				Message bus	Bridge connector
MUSDAC	X					Service desc.	Connection to middleware
INDISS/ Nemesys	X					Service desc.	Yes
uMiddle	X	X				Service desc.	Yes
OSDA	X					Service desc	Yes
SeDiM	X				X	Service desc	Yes
SATIN	X	X				Mobile components	Choice of SATIN for all
Jini		X				Mobile proxies	Choice of Jini for all
Semantic Middleware			X	X		Service desc. With semantic information	Choice of same semantic middleware for all
Semantic Web Services		X	X	X		WSDL Services	WSDL for all plus commitment on a semantic framework and ontologies

4 Architecture

In this section, we provide a first specification of the CONNECT architecture. This initial architecture reflects the CONNECT vision as identified in the project's Description of Work [1] and as reinforced during the first year of the project. Still, this first architecture does not reflect all the refinements elaborated in Work packages 2 to 5 and presented in Deliverables D2.1 to D5.1. Refining the architecture is an iterative process all along the project's duration, and this first refinement will be carried out in the next project's period. Nevertheless, first attempts at validating the initial architecture are already reported in Section 5, where the experiments carried out (some of them in collaboration with the other work packages) address some first refinement work on the architecture.

In our elaboration of the initial CONNECT architecture, we have opted for an approach that – at this step – makes the fewest possible assumptions and sets minimum constraints on the produced architecture. This has been motivated by the open, dynamic nature of the environments targeted by CONNECT, and by CONNECT's aim to make a breakthrough beyond technology and time barriers for enabling eternal systems.

In the following sections, we first provide an overall view of actors in the CONNECT open environment and architecture (Section 4.1); and introduce a set of models that model CONNECT actors and provide the basis for eliciting CONNECT architecture functionalities (Section 4.2). We then introduce the main phases of the CONNECT runtime (Section 4.3); and provide an integrated view of these phases, which constitutes the CONNECTed system life-cycle (Section 4.4). We finally discuss deployment considerations in the CONNECT open environment and architecture (Section 4.5), and conclude with a brief discussion (Section 4.6).

4.1 Overall View of Actors in the CONNECT Architecture

In this section, we provide a first specification of the CONNECT architecture. This initial architecture reflects the CONNECT vision as identified in the project's Description of Work [78] and as reinforced during the first year of the project. Still, this first architecture does not reflect all the refinements elaborated in Work packages 2 to 5 and presented in Deliverables D2.1 to D5.1. Refining the architecture is an iterative process all along the project's duration, and this first refinement will be carried out in the next project's period. Nevertheless, first attempts at validating the initial architecture are already reported in Section 5, where the experiments carried out (some of them in collaboration with the other work packages) address some first refinement work on the architecture.

In our elaboration of the initial CONNECT architecture, we have opted for an approach that – at this step – makes the fewest possible assumptions and sets minimum constraints on the produced architecture. This has been motivated by the open, dynamic nature of the environments targeted by CONNECT, and by CONNECT's aim to make a breakthrough beyond technology and time barriers for enabling eternal systems.

Accordingly, we identify the following *actors* in the CONNECT architecture:

- *Networked systems* are systems that manifest the will to connect to other systems for fulfilling some intent identified by their users and the applications executing upon them.
- *Enablers* are networked entities in the environment of networked systems that incorporate all the intelligence and logic offered by CONNECT for enabling connection between heterogeneous networked systems. Enablers constitute the *CONNECT enabling architecture*.
- *CONNECTors* are the emergent connectors produced by the action of enablers.
- *CONNECTed systems* are the outcome of successful creation and deployment of CONNECTors.

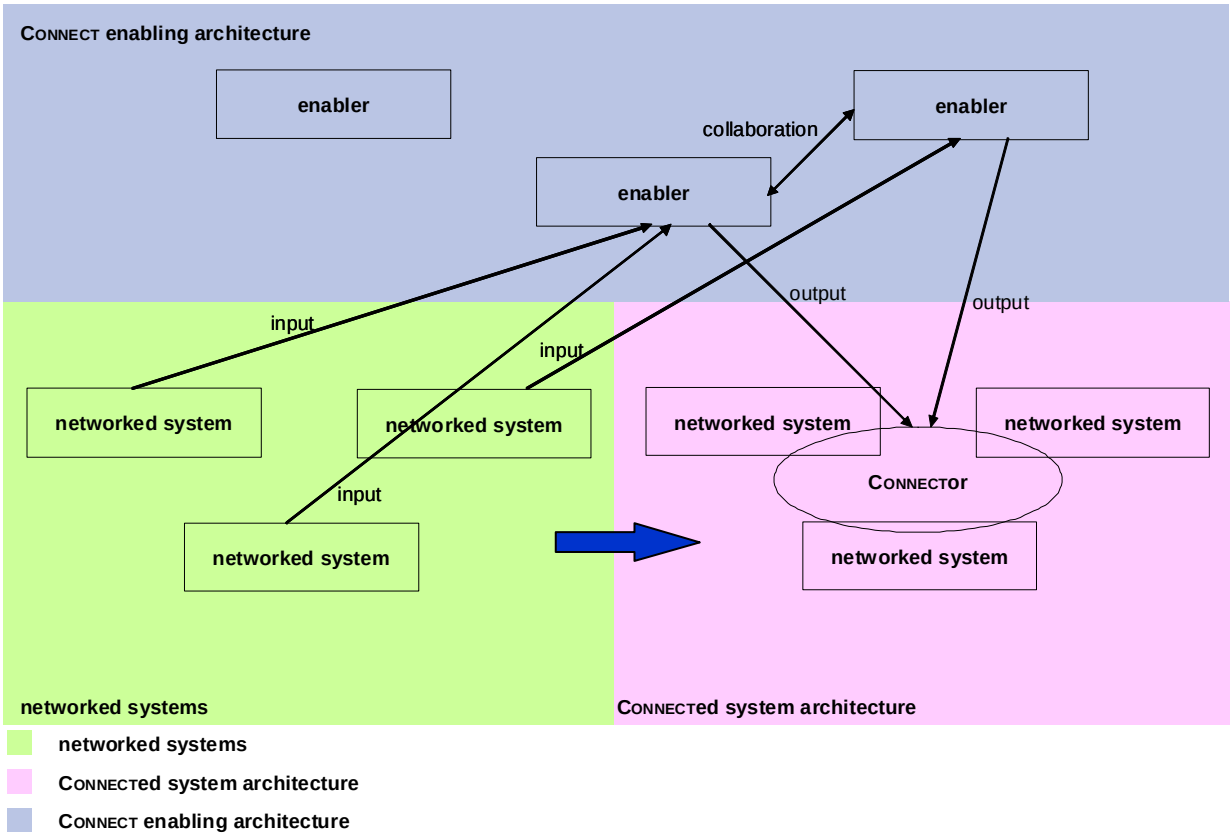


Figure 4.1: Actors in the CONNECT architecture

The actors behave as depicted in the high-level view of Figure 4.1. Networked systems manifest somehow their will to connect. This will, along with information about the networked systems, is communicated in the form of some input to the enablers. One or more enablers collaborate to synthesize and deploy a CONNECTor that enables networked systems to connect and fulfil their individual intents.

In Figure 4.1, distinct background colours are used for delimiting networked systems, the CONNECT enabling architecture, and the CONNECTed system architecture. We maintain the same colour convention in the following sections for qualifying architectural elements associated to one of these three domains.

4.2 CONNECT Models

Modelling is essential in the CONNECT architecture. Models enable: abstracting CONNECT actors and their features, describing them in a meaningful way, communicating description information among actors, inferring and completing – when incomplete – such information by observing actors, reasoning on such information, and finally synthesizing and deploying emergent actors, i.e., CONNECTors. CONNECT models aim at capturing architectural, behavioural, data and other dimensions within the CONNECT architecture.

In the following sections, we introduce modelling for networked systems (Section 4.2.1), sketch the CONNECT compositional connector model and algebra aiming at providing the basis to connector modelling and manipulation (Section 4.2.2), and provide a high-level model for CONNECTors (Section 4.2.3).

4.2.1 Networked System Model

Modelling networked systems is the first step towards enabling their manipulation (in the sense of assistance) by the CONNECT enabling architecture. We specifically aim at modelling their external interaction behaviour. We consider two levels of interaction, *middleware-layer interaction* and *application-layer interaction*. For lower layers below the middleware, we rely on widely established interoperability standards (e.g., IP for the network protocol), even if we assume that the middleware may possibly control lower network functionalities in a cross-layer fashion. Our networked system model is depicted in Figure 4.2.

Regarding middleware-layer interaction, we identify both *provided* (by the system) and *required* (from other systems) *behaviour*. Such behaviour is characterized by a set of features:

- *Process* specifies the supported interaction protocol, e.g., in the form of sequences of exchanged messages and states.
- *Coordination patterns* characterize the role semantics in an interaction, such as client-server or peer-to-peer, orchestration or choreography.
- *Interaction patterns* characterize the semantics of interaction protocols, such as message-based, event-based or shared-memory-based, synchronous or asynchronous.
- *Data transfer protocol* specifies the representation of data and control in the messages conveyed by the interaction protocol.
- *Addressing scheme* specifies the naming and referencing convention employed to uniquely identify a networked entity.
- *Data type system* specifies the representation of data types applied to all the data conveyed by the data transfer protocol.
- *Data* is the actual data information conveyed by the data transfer protocol.

Regarding application-layer interaction, we identify a subset of the features described above for the middleware-layer interaction. These features have similar meanings at the application layer. In general, applications rely on middleware for their external interaction, which means that they incorporate – more or less transparently – the semantics of the underlying middleware, and add their own semantics on top of that. For example, a process specifies the logic of an application more or less independently of the underlying middleware, and an application that uses shared memory can implement many different coordination and interaction patterns on top of this middleware.

Besides the above view of the application layer, we consider a second view where we focus on *application components*, their *intent* and their provided and required *functionalities*, while their interaction behaviour is here implicit. The essential feature here is the *interface*, that is, a description of the set of functionalities of the component made accessible to (but also required from) its environment. Typically, this description comes in the form of a set of data inputs and associated outputs following a specific data type system.

In addition to the above functional features, we are also interested in modelling *non-functional properties* of networked systems, which are orthogonal to the functional aspects. In our high-level view, we consider provided and required non-functional properties, their *description*, and their *enforcement*, that is, how they are functionally implemented by the networked system. Non-functional properties of interest to CONNECT are indicated in Section 4.3.4.

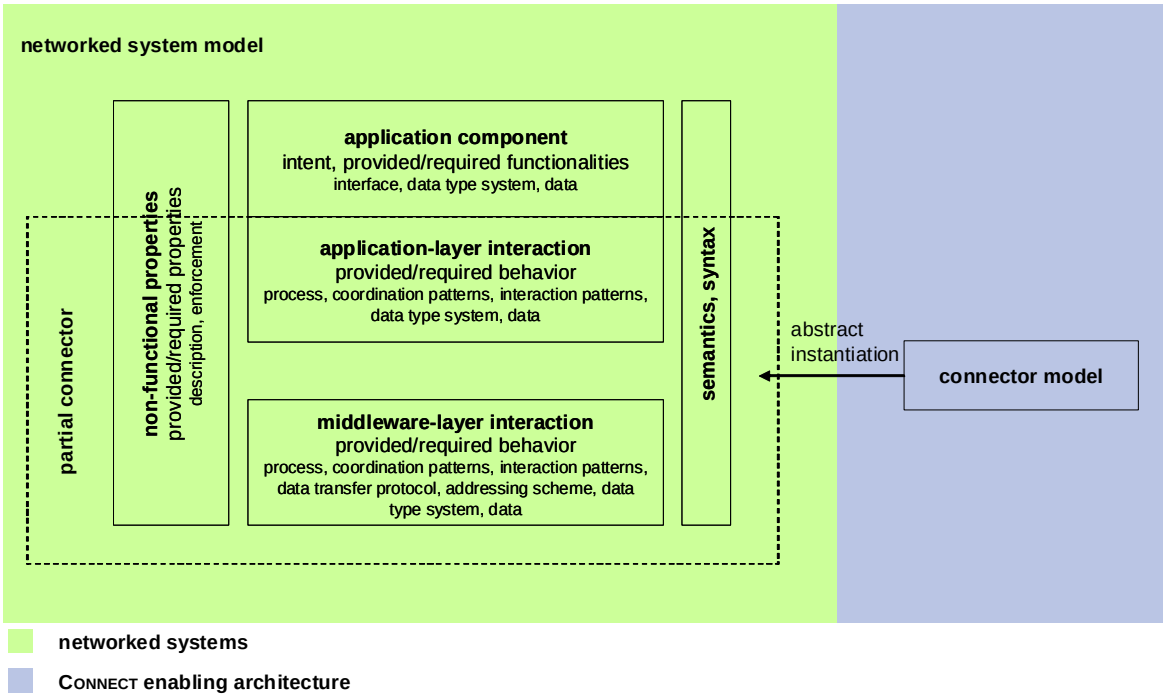


Figure 4.2: Networked system model

Finally, two factors applying to the modelling of all the above features are the *semantics*, i.e., the precise meaning, and the *syntax*, i.e., the representation used in the external interaction of the networked system.

Combined together, the two identified layers, that is application-layer interaction and middleware-layer interaction, provide a complete specification of the external interaction behaviour of a networked system, which we call *partial connector*, as it provides only the side of this networked system in a potential connection (modelled with a connector) with other networked systems. We consider that the partial connector is produced as an abstract instantiation of a more general connector model, which we introduce in the following section.

4.2.2 Compositional Connector Model and Algebra

A key objective in CONNECT is to establish a new formal model for describing connectors and an associated algebra that will support all aspects of connector manipulation. Figure 4.3 presents our high-level functional view of this model and algebra.

A connector is characterized by a set of functional attributes, such as: process, coordination patterns, interaction patterns, which take similar meanings to the ones identified in Section 4.2.1; data representation, relating to the syntactic representation of data within the connector; data manipulation, relating, e.g., to possible transformation of data by the connector. Many other properties may be added to this list depending of the nature of the connector, such as communication, coordination, conversion connector, etc. [79]. Furthermore, the meaning of non-functional properties, semantics and syntax in the connector model has already been identified in Section 4.2.1.

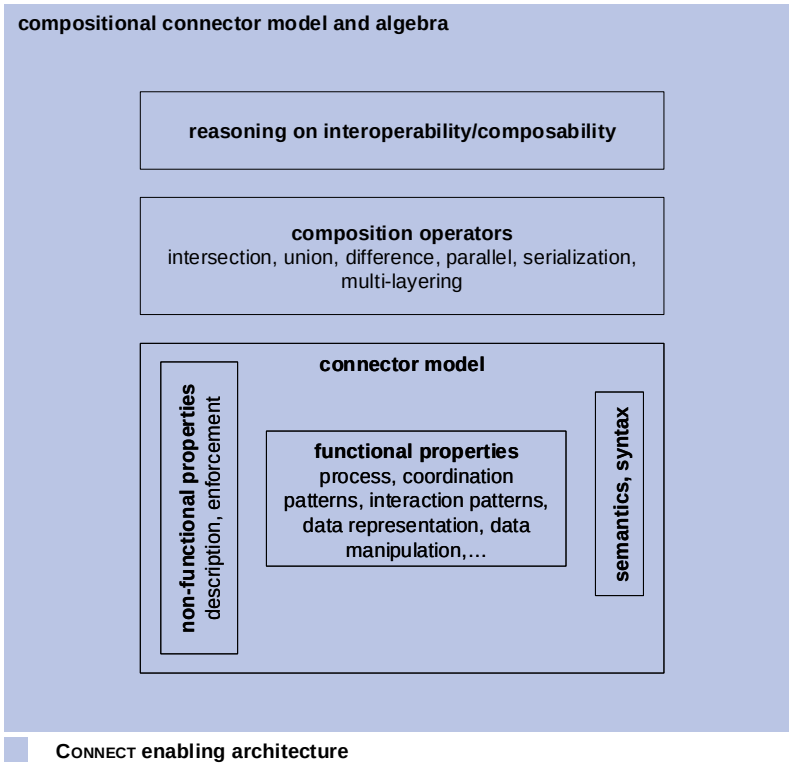


Figure 4.3: Connector model and algebra

On top of the connector model, we provide a very high-level view of the associated algebra. A key property sought for connectors in CONNECT is compositionality, which will enable combining connectors but also factorizing connectors and reusing connector subparts. Incrementality and reuse are essential for establishing a knowledge base of (previously synthesized and successfully used) connectors and facilitating their automated and on-the-fly manipulation in an efficient manner. We identify a number of *composition operators* to be applied on connectors, such as intersection, union, difference, parallel, serialization, multi-layering.

Finally, relying on the connector model and the composition operators, the CONNECT algebra will enable reasoning on composability of connectors and interoperability provided by them to heterogeneous networked systems, and will provide the basis to synthesizing emergent connectors.

As may be seen in the above, the aim of the CONNECT connector model and algebra is to enable modelling and manipulation of both partial connectors of networked systems and CONNECTors. The relation between the two is clarified in the following section.

4.2.3 CONNECTor Model

As illustrated by Figure 4.4, partial connectors, already deployed by networked systems, later make part of the CONNECTor in the CONNECTed system architecture. More specifically, partial connectors provide each a *partial view of the CONNECTor* as seen by the networked system owning the specific partial connector. This abstraction leads us to establish the CONNECTor model as depicted in Figure 4.5.

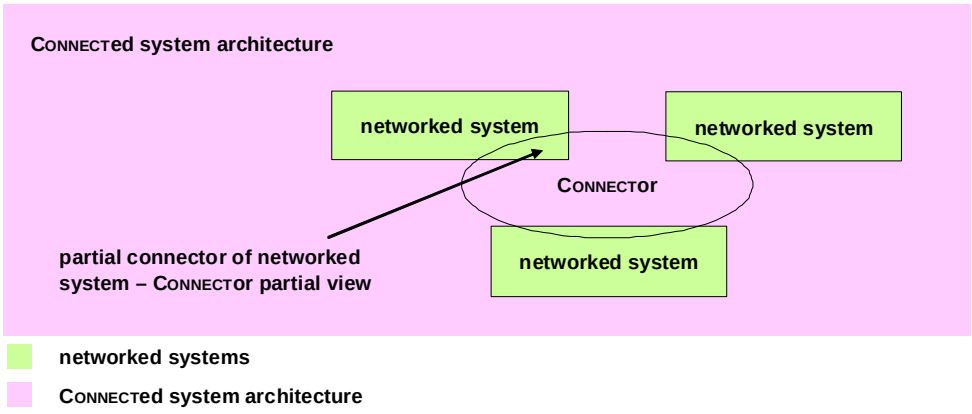


Figure 4.4: Partial connectors vs. CONNECTor

In this model, the CONNECTor incorporates all the partial views of the wished interaction as carried by the individual partial connectors, to which are linked the application components of the networked systems. The CONNECTor internal logic then provides a *unification of partial views* that enables bridging of the heterogeneous networked systems. Functions such as mapping, translation, mismatch resolution, refinement, substitution and bridging characterize this unification. Furthermore, besides functional unification, the CONNECTor ensures non-functional unification by reconciling and enforcing end-to-end non-functional properties on the CONNECTed system based on partial properties of the networked systems.

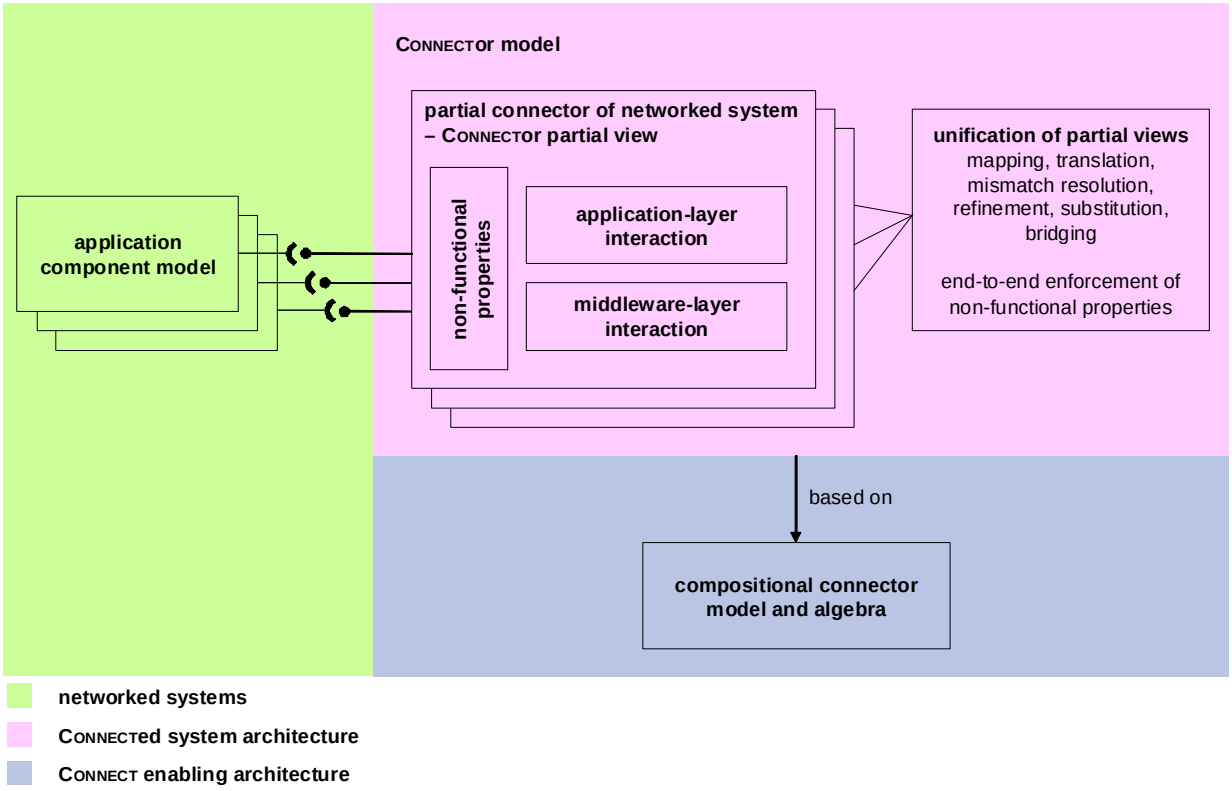


Figure 4.5 CONNECTor model

The CONNECTor model and related synthesis of CONNECTors as viewed herein will rely on the compositional connector model and algebra sketched in Section 4.2.2.

4.3 Main phases of the CONNECT Runtime

Targeting open, dynamic and uncoordinated (i.e., bearing no static design of coordination between actors) environments, CONNECT exhibits all its functionalities at system runtime. In accordance with the overall view of the CONNECT open environment and architecture introduced in Section 4.1, we identify the following phases of the CONNECT runtime:

- *Discovery* enables networked systems to manifest their will to connect to other networked systems and to discover mutually interested networked systems, while at the same time allows the CONNECT enabling architecture to retrieve initial information on likely-to-be-associated networked systems (Section 4.3.1).
- *Learning* is performed by enablers upon networked systems for completing the initial information about the latter provided by discovery. The outcome of combined discovery and learning should be a sufficiently good view of the partial connectors owned by the networked systems (Section 4.3.2).
- *Synthesis* is performed by enablers for generating and deploying an appropriate CONNECTor that will successfully bridge the heterogeneous partial connectors and establish a CONNECTed system (Section 4.3.3).
- *Verification & validation* is performed by enablers during and after the synthesis phase for ensuring the correctness of the CONNECTor and the running CONNECTed system with respect to the requirements and intents of the involved networked systems (Section 4.3.4).

In the following sections, we discuss in more detail the above identified phases of the CONNECT runtime as actions on the models introduced in Section 4.2.

4.3.1 Discovery

Runtime discovery is a typical activity performed in open dynamic environments, where actors need to be mutually discovered and identified before actually connecting and interacting. A basic assumption made by CONNECT is that networked systems are discovery-enabled. Following the typical runtime discovery paradigm, discovery in CONNECT is illustrated in Figure 4.6.

In a first step towards analyzing CONNECT discovery, we assume that networked systems are abstracted as *instantiations of the networked system model* introduced in Section 4.2.1. Such instantiated model provides a complete description of the networked system.

However, only a subset of this complete description is made available by the networked system to its environment via discovery. This is due to the fact that networked systems are commonly designed to interact within their technological island (see Section 4.1), where a great part of information concerning the networked systems is common and taken for granted, and thus there is no need to be externalized. We call the externalized information subset *a priori description* of the networked systems. This a priori description may concern (part of) the intent and functionalities of application components, application- and/or middleware-layer behaviour, and non-functional properties. We assume that the a priori description conveys, besides syntactic information, also semantic information about the externalized networked system features. This semantic information is necessary in open environments, where semantics cannot be assumed to be inherently carried in a commonly agreed syntax. Typically, ontologies are used in open environments for providing a common vocabulary on which semantic descriptions of networked systems can be based.

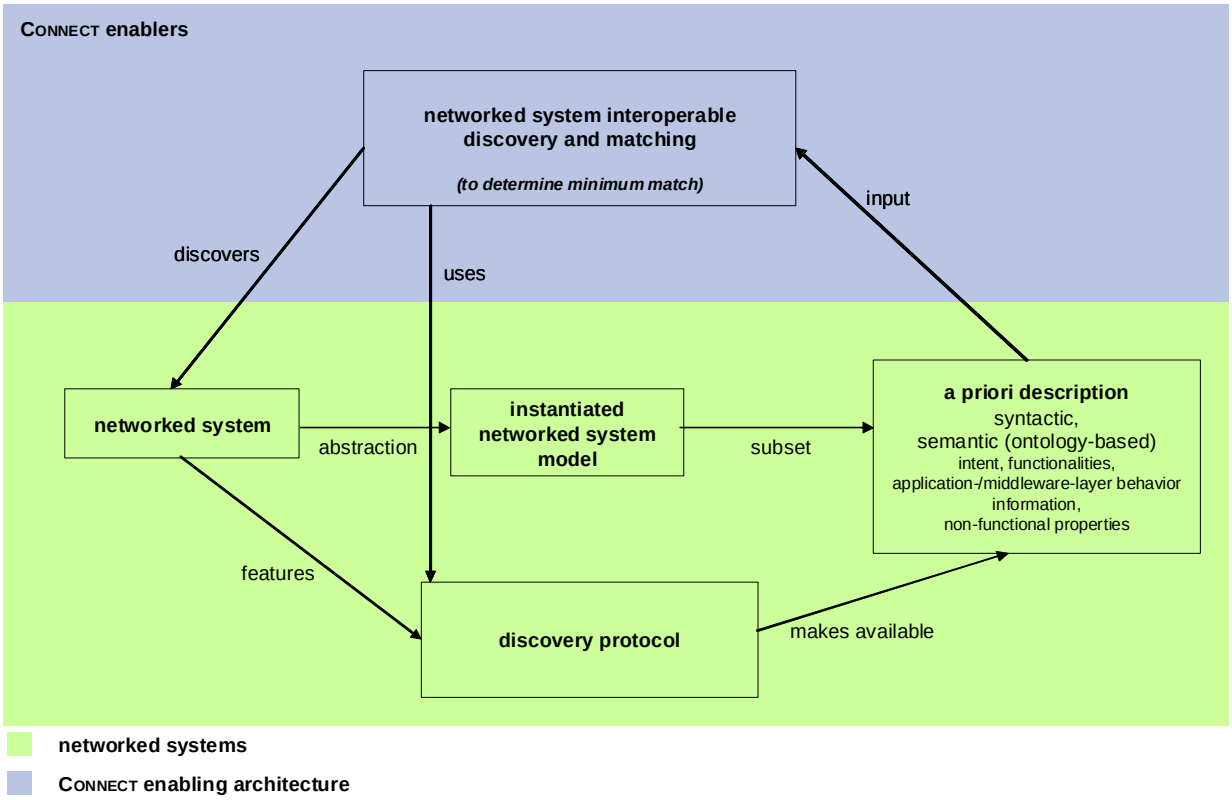


Figure 4.6: CONNECT Discovery

The a priori description of a networked system is made available to its environment via a *discovery protocol* featured by the networked system. Networked systems discover each other by using such discovery protocols. In the same way, CONNECT enablers can discover networked systems by using their discovery protocol and thus retrieve their a priori descriptions. This discovery naturally follows the semantics of the employed discovery protocol; it can thus be active, passive or both ways.

Performing in an open, heterogeneous environment, CONNECT discovery should be able to make use of *any available discovery protocol* employed by the networked systems. In the same way, discovery should be able to *exploit heterogeneous a priori descriptions* exposed by the networked systems. Thus, interoperable discovery is a key issue in CONNECT. The objective of CONNECT discovery is to identify networked systems that are likely to be associated; this is typically performed by matching based on the a priori descriptions.

Existing discovery approaches for open environments already propose flexible, non-exact matching between networked systems that can handle limited syntactic and semantic mismatch in the functionalities of application components. However, in CONNECT, mismatch can be much more significant and may concern any feature of the networked systems. Thus, a key issue of CONNECT matching is to *determine the minimum match* between networked systems that ensures the feasibility of bridging them via CONNECT. Determining the minimum match depends not only on the networked systems in hand but also on the capacity of CONNECT (see next phases in the CONNECT runtime) to handle the corresponding mismatch.

4.3.2 Learning

Learning of networked systems is performed just after discovery and successful matching, and is necessary due to the fact that the retrieved a priori descriptions of networked systems are incomplete, as discussed in Section 4.3.1.

Learning acts on the same networked system models as discovery, as depicted in Figure 4.7.

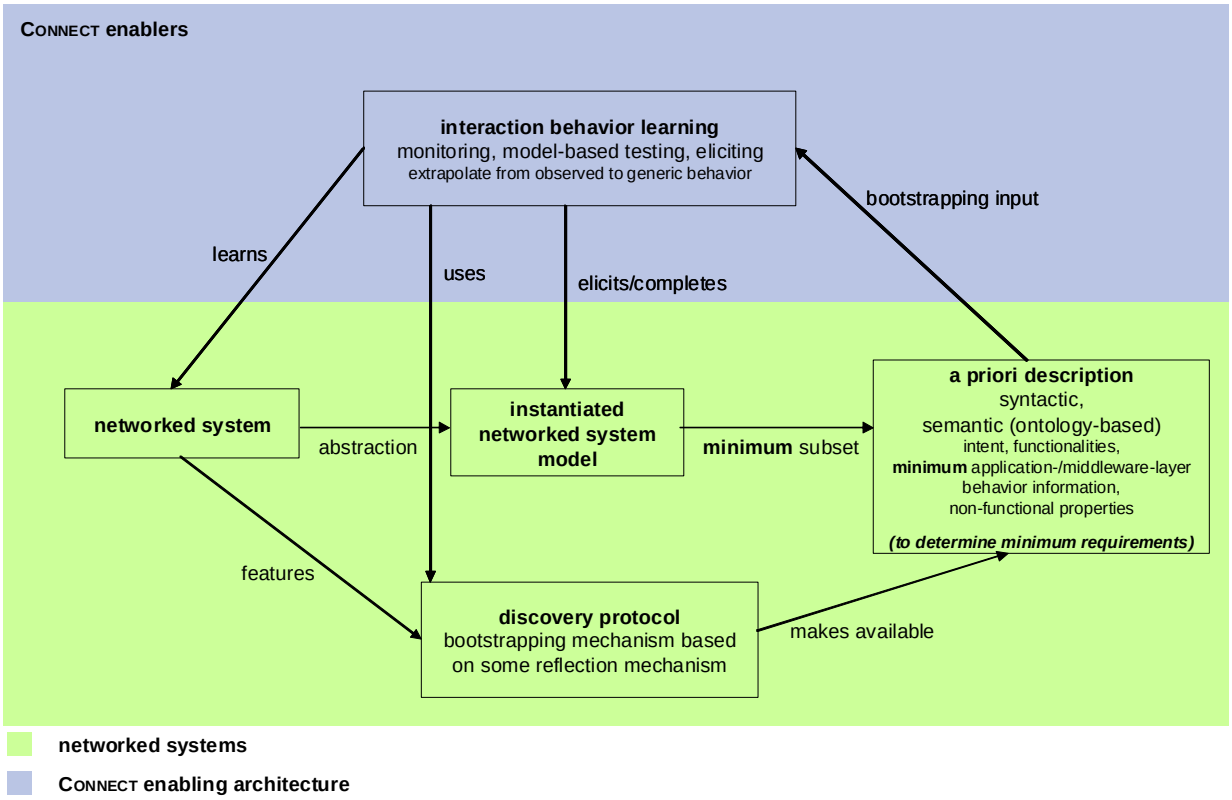


Figure 4.7: CONNECT Learning

Learning needs an a priori description of networked systems that it will then attempt to learn. CONNECT learning focuses on the interaction behaviour of networked systems. We assume that information is available on the intent and functionalities of application components, typically in the form of an interface; and that minimum information is possibly also available on application- and middleware-layer behaviour. Then, learning attempts to infer the complete interaction behaviour. We still need to determine the minimum requirements that learning has with regard to the a priori description; accordingly, we need to determine the minimum subset of the instantiated networked system model that should be included in the a priori description.

Learning views the discovery protocol in a general way as a bootstrapping mechanism, possibly based on some reflection capacities featured by the networked systems, which enables launching the learning process. The a priori description made available by the discovery protocol is exactly the bootstrapping input that learning needs.

CONNECT learning employs methods based on monitoring and model-based testing of the networked systems to elicit their interaction behaviour. Learning attempts to extrapolate from observed behaviour to generic behaviour. The outcome of learning is a complete – as far as possible – instantiated networked system model.

4.3.3 Synthesis

Synthesis receives the output of learning and attempts to synthesize a CONNECTor adapted to the networked systems in hand and successfully resolving their incompatibilities. Figure 4.8 depicts CONNECT synthesis.

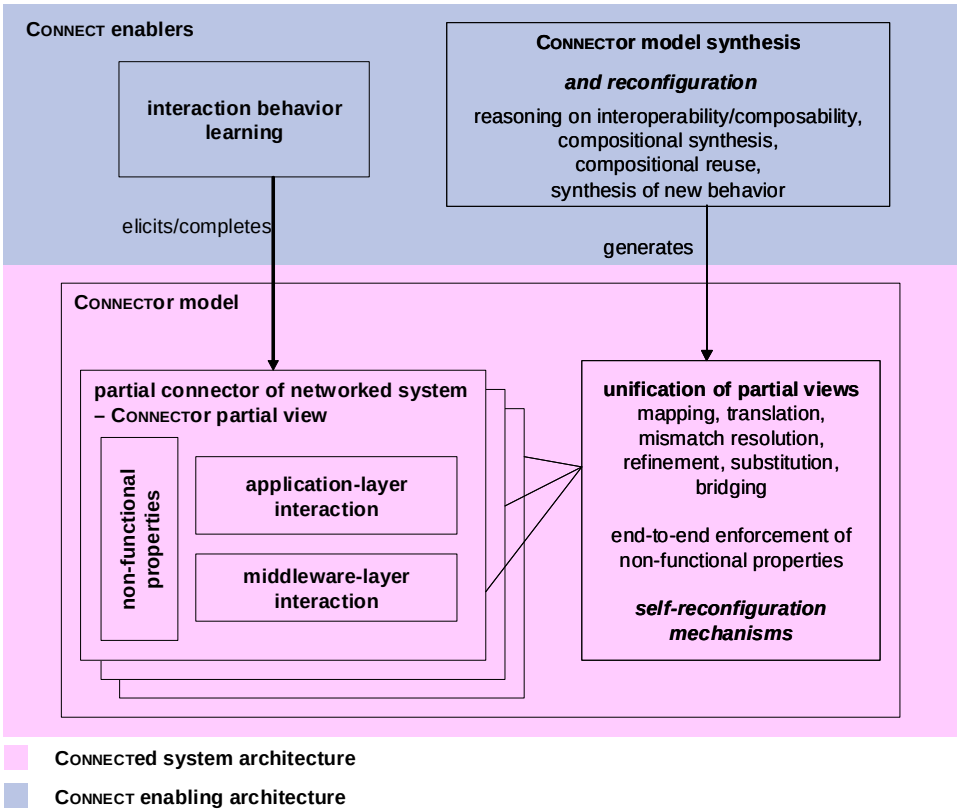


Figure 4.8: CONNECT Synthesis

From the synthesis point of view, what learning actually provides are models of the partial connectors of the networked systems involved, that is, partial views of the CONNECTor to be synthesized. Synthesis then devises a unification of these partial views integrated into the logic of the CONNECTor.

Synthesis acts at the level of the CONNECTor model and combines: reasoning on interoperability and composability of partial connectors, compositional synthesis of new connectors, compositional reuse of existing connectors and connector subparts, and synthesis of new behaviour integrated into new connectors. Efficiency of the synthesis process is a key issue and will be heavily supported by the compositional characteristics of the connector model and algebra outlined in Section 4.2.2.

Furthermore, targeting eternal systems that may change and perform in a changing environment, but also recognizing the fact that models of networked systems may also be updated due to a better understanding of the systems in time, CONNECT enables reconfiguration of the CONNECTor model at two levels. First, the generated CONNECTor model integrates self-reconfiguration mechanisms able to respond to changes in the environment in which the CONNECTor will execute. Second, a major reconfiguration may be needed, which then is undertaken by the synthesis process; efficient reconfiguration (in the same way as for the initial synthesis) is enabled by the CONNECT compositional connector model and algebra.

The next steps of the synthesis process, after the synthesis of the CONNECTor model, concern producing and deploying a real executable CONNECTor, thus establishing a CONNECTed system. These steps are intermingled with the verification & validation process, as discussed in the following section.

4.3.4 Verification & Validation

Verification & validation (V&V) concerns checking the CONNECTor during the last steps of its synthesis (after the generation of the CONNECTor model and until its deployment that

accomplishes a CONNECTed system), but also checking the executing CONNECTed system during its lifetime.

The steps taken by the combined synthesis and V&V process are depicted in Figure 4.9.

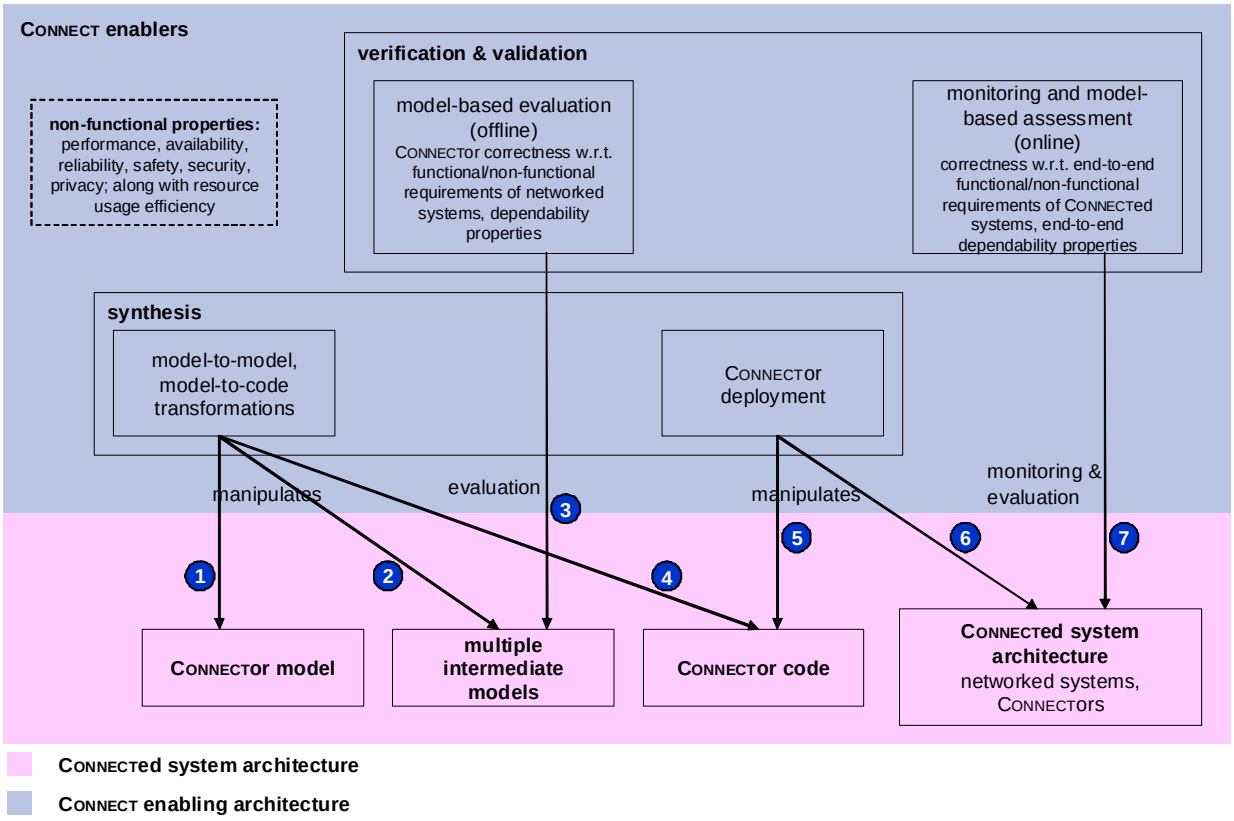


Figure 4.9: Combined CONNECT Synthesis and V&V

In Steps 1 and 2, synthesis performs a number of model-to-model transformations to produce multiple intermediate models from the CONNECTor model. These models serve two purposes: first, they make part of a model-driven engineering process with the final objective to produce executable code for the CONNECTor; second, they are used by the V&V process.

In Step 3, V&V performs an offline (i.e., before it is put into operation) evaluation of the CONNECTor in hand based on the models produced in the previous steps. This evaluation aims at checking the correctness of the CONNECTor with respect to the functional and non-functional requirements of the networked systems including certain dependability properties.

In Step 4, and after the successful evaluation of the CONNECTor, synthesis performs a final model-to-code transformation to produce executable code for the CONNECTor.

In Steps 5 and 6, synthesis deploys the CONNECTor code and produces an actual CONNECTed system comprising the networked systems and the CONNECTor. A principal concern is about where the CONNECTor will be deployed and run: enablers will normally be able to host the executing CONNECTor; other solutions are possible. We provide an initial discussion of deployment issues in the CONNECT architecture in Section 4.5.

Finally, in Step 7, V&V performs an online evaluation of the CONNECTed system during its actual execution, which relies on monitoring and model-based assessment. Models of the CONNECTed system are produced from models of the CONNECTor and the networked systems available from the present and the previous phases of the CONNECT runtime. This online evaluation aims at checking the correctness of the CONNECTed system with respect to end-to-end functional and non-functional requirements including end-to-end dependability properties. V&V reports on the CONNECTed system during its lifetime, thus providing feedback to previous

phases of the CONNECT runtime, as we will further discuss in the next section; this is a key capacity of CONNECT towards enabling eternal systems.

As already discussed for V&V but also in the other phases of the CONNECT runtime, non-functional properties of networked systems, of CONNECTors and finally of CONNECTed systems are essential in the CONNECT vision. Non-functional properties of interest to CONNECT include performance, availability, reliability, safety, security, and privacy; resource usage efficiency is another key property, as resource-constrained actors will be the norm in the open dynamic CONNECT environment.

4.4 Integrated View of the CONNECT Runtime: CONNECTed System Lifecycle

This section provides the complete view of the CONNECT continuous runtime process with respect to the CONNECTed system lifecycle, which aims at enabling eternal systems, i.e., ensuring system connectivity in a changing environment (Figure 4.10).

Triggered by one or more networked systems manifesting their will to connect, *interoperable discovery and matching* identifies the networked systems that are likely to be associated based on their a priori descriptions, and communicates this information to *learning* (Step 1). Learning infers the interaction behaviour of the identified networked systems and completes their a priori descriptions, thus eliciting – as precisely as possible – their partial connector models, which it feeds into *synthesis* (Step 2). Synthesis generates a CONNECTor model able to bridge the heterogeneous partial connector models (Step 3). By successive *model-to-model transformations* (Step 4), synthesis generates appropriate models of the CONNECTor required by *verification & validation (V&V)* (Step 5). V&V evaluates the CONNECTor offline and provides feedback to synthesis, which may lead to *reconfiguration* or *resynthesis* of the CONNECTor model (Step 6). At the end of this synthesis and evaluation cycle, synthesis performs a *model-to-code transformation* to generate an executable CONNECTor (Step 7). Synthesis then *deploys* the CONNECTor code accomplishing a CONNECTed system, which *executes*; during this execution, the CONNECTor is able to *self-reconfigure* to respond to changes in its environment (Step 8). All along the CONNECTed system execution, V&V monitors and evaluates it (including networked systems and CONNECTor) online (Steps 9, 10). At the same time, learning also monitors the CONNECTed system to update and improve its learned models of the constituent networked systems (specifically, V&V and learning apply similar monitoring and model-based testing mechanisms) (Step 11). An evaluation of the networked systems or an update of the learned networked system models will be shared between V&V and learning (Step 12). In case of negative evaluation of the CONNECTed system, or some problem detected during its execution, or some significant update of the learned networked system models, or some change of the networked systems or their environment, V&V provides feedback to synthesis triggering a resynthesis of the CONNECTor model and consequently re-execution of all the steps that follow (Step 13). This puts in place a continuous CONNECT process that takes account of change and evolution towards enabling eternal systems.

As already pointed out throughout this chapter, models are essential to the CONNECT process. Models are communicated, learned, synthesized, transformed, verified and reconfigured throughout the CONNECTed system lifecycle. Models will be compositional and reusable. Such models constitute the shared knowledge among CONNECT actors in the CONNECT environment. Accumulating and reusing this knowledge is essential in CONNECT. Reusing solutions or parts of solutions greatly contributes to efficiency. But also, accumulated knowledge is aggregated experience in ensuring interoperability between heterogeneous systems. This is further related to the CONNECT vision of enabling eternal systems, systems that can connect to any existing but also to future systems. Hence, a *knowledge base* of models and operations on models is envisioned in CONNECT, accessible and shared among CONNECT enablers. Such base will contain compositional models of connectors and connector patterns and related operators,

enabling: factorization and (partial) reuse of models, syntheses and transformations; and incremental update of models.

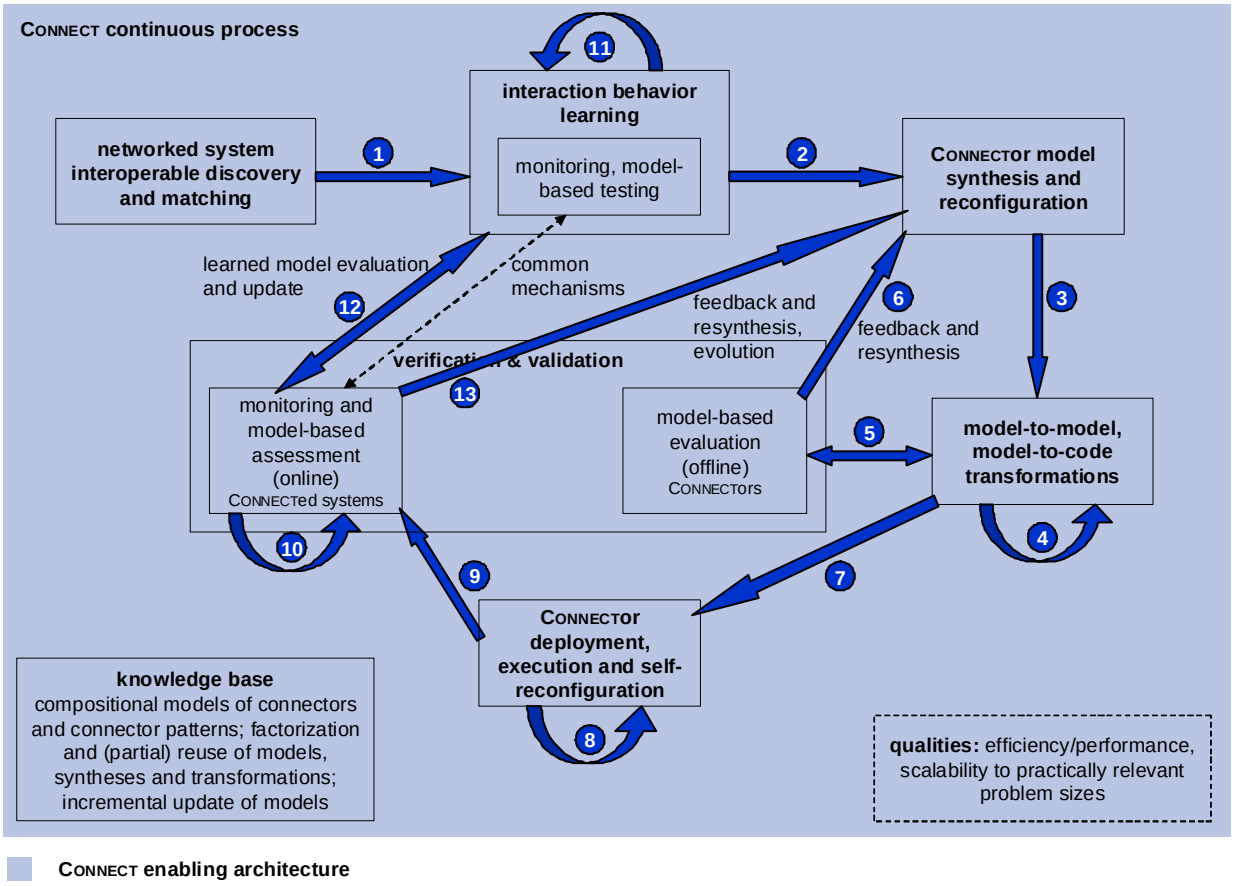


Figure 4.10: CONNECT continuous process and CONNECTed system lifecycle

Last but not least, aiming at real-life, practical applications, the CONNECT process as a whole should ensure three important and interrelated qualities: *performance* adequate for dynamic, often user-interactive environments; *efficiency* with respect to available – often constrained – resources; and *scalability* – in particular of applied formal models and methods – to practically relevant problem sizes.

4.5 Deployment Considerations in the CONNECT Open Environment

Throughout this chapter, we introduced the CONNECT architecture and actors and the relations between them. We provided in Figure 4.1 a high-level view of the CONNECT open environment making minimum assumptions about the actual functional deployment of the CONNECT architecture. Furthermore, we discussed non-functional properties of interest to CONNECT applying to networked systems, CONNECTors and finally CONNECTed systems.

However, as the CONNECT enabling architecture directly intervenes in the interaction between networked systems, i.e., in the functioning of CONNECTed systems, it has an impact on the functional and non-functional properties of CONNECTed systems. Our objective in CONNECT is to minimize this impact, i.e., CONNECT intervention should be as transparent as possible to networked systems, while ensuring (and reconciling) their functional and non-functional properties and requirements within the CONNECTed systems.

Related to the above, key issues in the relations between all actors in the CONNECT open environment (networked systems, enablers, CONNECTors) are *security*, *privacy* and *trust*.

Security and privacy are non-functional requirements not only in the interactions between networked systems, but also in their relations with the enablers and in their reliance on an external entity (the CONNECTor) provided by the enablers. In the CONNECT open environment, where actors are associated dynamically and have no previous acquaintance of each other, trust will play a principal role. Hence, besides considering security and privacy as some of non-functional requirements of networked systems to be taken into account in the CONNECT process (as already discussed in the previous sections), a specialized architecture for security, privacy and trust should make part of the overall CONNECT architecture.

Furthermore, all the above considerations are directly related to the deployment of the CONNECT architecture. Deployment should be flexible to be able to deal with different situations in the CONNECT open environment. For example, a generated CONNECTor may be hosted by an enabler, deployed on one of the associated networked systems, or distributed among all the associated networked systems. In general, the distribution of the CONNECT architecture will depend on: available and required functionalities of actors; efficiency in performing the CONNECT process and in the actual execution of the resulting CONNECTed system; resource availability and management of actors; evolution of the CONNECTed system and its environment, and capacity to deal with evolution; security, privacy and trust between all actors involved.

4.6 Summary

We have introduced in this chapter the initial CONNECT architecture as an ensemble of actors, models, functionalities, and a continuous runtime process towards ensuring eternal CONNECTed systems. This is an abstract architecture, as targeted in the first project year, attempting at this step to set minimum constraints, thus aiming at connectivity beyond technology and time barriers.

The CONNECT architecture will be refined in a series of steps. Particularly, our approach will progress along two complementary axes.

- First, we will refine the CONNECT architecture with respect to the functionalities and capacities of the CONNECT enabling architecture, as these will become more concrete over the duration of the project. Related to this is the refinement of models, coordination between different models, and coordination between the phases of the CONNECT process.
- Second, we will elaborate on global issues concerning the CONNECT architecture, such as the roles of actors, the coordination between actors, the distribution and deployment of the architecture, and realization of the architecture towards enabling viable CONNECTed systems. In this latter effort, we will particularly consider different distributed environments, paradigms and current technologies, and identify the conditions and challenges of implementing CONNECT in real-world environments.

The evolving CONNECT architecture will serve as the coordinating element for all specialized work undertaken in parallel by the work packages WP2-WP5 but also WP1. Input to these WPs and feedback from them, as part of constant collaboration and exchange, will enable synchronising the global view with the specialized views within the project.

5 Experiments

In this section we present five experiments that focus on particular aspects of the CONNECT architecture, with the purpose of validating the initial architecture and identifying the important challenges to be resolved in future refinements of the architecture. These experiments are:

1) The *Popcorn Scenario Dry Run* experiment (Section 5.1) takes a top-down view of the CONNECT architecture and applies it to a single application scenario in order to validate the integration of the roles within the CONNECT process.
2) The *Implementing* CONNECT*ors* experiment (Section 5.2) takes a bottom-up view of the CONNECT architecture. Specific CONNECTors are implemented and deployed, and the constraints from real-world systems are identified.
3) The *Resolving Data Mismatches* experiment (Section 5.3) tackles the data interoperability problem using one ontology to resolve data mismatches across heterogeneous applications.
4) The *Ontology Heterogeneity* experiment (Section 5.4) analyzes the problem of ontology heterogeneity, i.e. we remove the assumption that one ontology is utilized to address the data mismatch problem, but rather we assume that different services refer to different ontologies. The objective is to evaluate to what extent state of the art tools are able to build bridge rules across ontologies.
5) The *CONNECTor Composition and Reuse* experiment (Section 5.5) investigates the performance, deployment gains, and challenges of considering re-use at the level of the CONNECTor elements.

5.1 The Popcorn Scenario Dry Run: A top-down approach to the CONNECT architecture

5.1.1 Goals

As introduced in Section 4, the goal of the overall CONNECT process is to take the information about the networked systems (NSs), and generate CONNECTors so that they can communicate and form a CONNECTed system. In this experiment we perform a top-down method to produce an initial idea of what the results of the overall CONNECT process will look like; this combines: i) the requirements of the project as outlined in [1], ii) input from the individual work packages, and iii) the expertise of the project partners.

To enrich this experiment we took one of the scenarios (the popcorn seller application discussed in Section 2 of this report) and performed a dry run of the CONNECT runtime process introduced in Section 4.4. This eschewed implementation and examined how the CONNECT enablers would behave when connecting this application in terms of ‘paper’ inputs and outputs only.

The general aims of the experiment were:

- To foster early integration of all project partners in developing the CONNECT solutions; to achieve such universal interoperability, integration of the individual areas of expertise and technologies is fundamental.
- To present an initial view of the CONNECT system workflow in terms of exchange of information from enabler to enabler, by building upon and refining the system view given in Section 4.
- To offer an early validation through a common use-case of the initial CONNECT architecture.

This section presents a high-level view of the experiment and procedures of the dry run[4]. Individual work packages, i.e., WP2-WP5, have produced more specific results in terms of their individual enablers within the scope of this experiment; we do not report all of this information here and instead we point to these results within this section.

The experiments remain ongoing and we intend to use the lessons learnt to further improve the design of and the coordination between the CONNECT enablers.

5.1.2 Methodology and Results

The first step to identifying a high-level view of the CONNECT operation was to identify the individual activities that work together to provide the behaviour of the CONNECT architecture, and how these work together. This was achieved by eliciting the information from the original description of work, as well as by referring to the CONNECT architecture discussed in Section 4.

Stage 1: A High Level View of CONNECT Operation

The activities performed at various phases of the CONNECT runtime process were extracted from the description of the CONNECT architecture discussed earlier, and this list was further refined by discussions among the project participants. These include discovery, learning, synthesis, code generation, monitoring, dependability analysis, security enforcement, and trust management. The summary of these activities are provided in Section 4, and their details are available in the deliverables of the work packages responsible for them. In this stage of the experiment, the partners of the CONNECT project were asked to clearly identify the activities that they were in-charge of, and this information was used in Stages 2 and 3.

Stage 2: Information Flow in the CONNECT Process

The operation of the CONNECT system is made possible by the activities of various enablers, which exchange information about the networked systems in order to construct and deploy the CONNECTors needed for the CONNECTed system. In this step we asked each work package to provide the following information about the particular activities they were involved with:

- Input Requirements: What information do you need as input to your activity, and what language do you need it described in?
- Output Details: What information is included in the output of your activity, and what language will this output be in?

This information was analysed and an overall flow of information was produced that refined the high-level process of Section 4.4 to describe an initial view of how this will operate in practice. Table 5.1 documents the inputs and outputs that flow between the activities of the process.

[4] Details available at http://www-roc.inria.fr/connect/connect-dry-run

Table 5.1: Information exchanged during the CONNECT Process

Information	Producer	Consumer
Interface description of networked system	Discovery	Learning
Ontology of data used in networked system	Discovery	Learning
Ontology mapping between message sequences observed in the NS	Discovery	Synthesis
Metrics of interest	Discovery	Learning, Dependability Analysis
Security policy of networked systems	Discovery	Security Enforcement, Synthesis
Guarantees provided by networked systems	Learning	Monitoring, Dependability Analysis
Formal description of behavior of networked systems	Learning	Synthesis, Dependability Analysis, Trust Management
Formal description of behavior of CONNECTor	Synthesis	Dependability Analysis, Trust Management, Code Generation
CONNECTor security contract	Synthesis	Security Enforcement
Expected distribution of metric values	Dependability Analysis	Synthesis
Trust assessment	Trust Management	Synthesis, Security Enforcement
Guarantees and conditions to check in the CONNECTed system	Security Enforcement, Learning, Synthesis	Dependability Analysis, Monitoring
Measurements and logs that match specified conditions	Monitoring	Learning, Synthesis, Security Enforcement, Trust Management
Code of CONNECTor	Code Generation	<used for deployment>

Using this information, the high-level view of the CONNECT system operation initially introduced in Figure 4.10 was refined to produce Figure 5.1; note that the diagram also identifies which work package is in-charge of each activity performed within the CONNECT process. Based on Figure 5.1, the overall workflow in the CONNECT process is as follows. Note that the numbers in the parentheses refer to the numbers denoting production or consumption of information in Figure 5.1.

I. Discovery (1) gathers information for each networked system and (2) provides the interface description, message sequence and data-domain ontologies, metrics of interest,and security policy to be used by other activies.

II. Learning (3) consumes the interface definitions, data-domain ontology, and metrics of interest to (4) actively interact with each NS and (5) produce a formal model of the behavior of the NSs (in Mealy machine form, later converted to LTS), and initial estimates about the guarantees provided by each networked system (e.g., “System x is seen to respond in no later than 10ms during the active learning process”). Note that the “quality” of these guarantees may depend on the exact techniques used to learn them, as well as the time available for the active learning process.

III. Synthesis then (6) takes the formal behavior models of the NSs, and their message sequence ontologies *properly mapped* to (7) produce a formal model of the CONNECTor, as well as guarantees conditions that should be matched by the CONNECTor.

IV. The formal models of both the NSs as well as the CONNECTor are then (8) combined into a dependability model of the CONNECTed system which is then (9) used in conjunction with the metrics of interest, provided guarantees, and conditions to check for (10) producing expected distribution of metrics by Dependability Analysis.

V. Trust Management (9) uses the dependability model to (11) produce a trust assement of the proposed CONNECTor.

VI. Synthesis then (12) reviews the expected distribution of metrics and the trust assessment values to confirm if the CONNECTor is suitable to be deployed. If not, steps (7) – (12) repeat. Ultimately, when the synthesized CONNECTor is found to be suitable, Synthesis instructs Code Generation to (13) use the formal model of the CONNECTor to (14) produce the code of the CONNECTor.

VII. Additionally, Synthesis also (15) uses the security policy of the NSs to (16) produce the CONNECTor security contract, which is then (17) used in combination with the trust assessment of the proposed CONNECTor as well as the security policies of the NS by Security Enforcement to (18) produce the security conditions to be checked in the CONNECTed system, as well as to (19) configure the hosts in the (soon to be) CONNECTed system for deployment of the CONNECTor.

VIII. Finally, at the end of the CONNECTor construction phase, the CONNECTor code is (20) deployed on the system, making it a CONNECTed system.

IX. After deployment, Monitoring (21) uses the metrics of interest and the guarantees and conditions to check in the CONNECTed system to (22) passively monitor the CONNECTed System and (23) produce measurements and logs that match the specified conditions. These measurements and logs are then (24) used by Learning, Synthesis, Security Enforcement, and Trust Management to further fine-tune their operations.

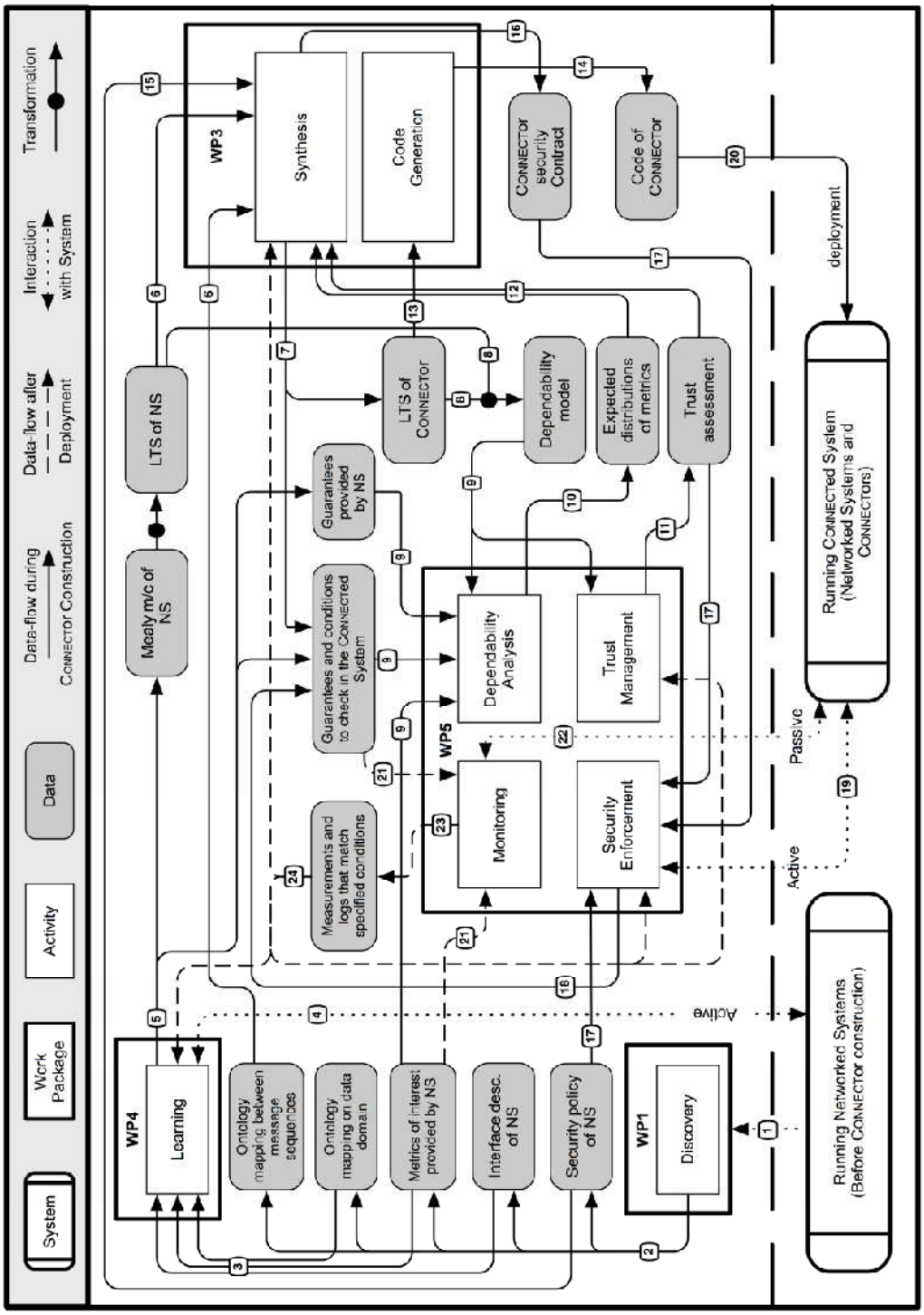


Figure 5.1 Data Flow in the CONNECT System

Stage 3: Validation using the 'Popcorn Dry Run'

In order to better understand the challenges encountered during the CONNECT process, we undertook an exercise where we followed the steps followed by the activities performed by some of the enablers (focussing more on the CONNECTor construction phase) while trying to ensure interoperability between a German Consumer and a French Merchant from Table 2.1 in Section 2. Here is a brief overview of the process which was followed:

1. **Discovery:** WP1 participants acted as the discovery enabler. Their role was to specify the interfaces that would be provided by the specific networked systems, namely the French merchant (using SSDP for advertising and SOAP for interactions) and the German consumer (using the Lime tuple space middleware). The output of the discovery enabler was then produced in the form of WSDL files (available at the dry-run website) for the description of the networked systems: i) the French merchant, and ii) the German consumer.
2. **Learning:** WP4 participants acted as the learning enablers, taking as input the description of the networked systems as provided by the WP1 participants acting as the discovery enabler. In addition, the WP4 participants simulated the learning process by hand-generating the responses which would be generated by the systems as they are subjected to active learning. Using the techniques further detailed in Section 2.3 of D4.1, first the mealy machine equivalents of the individual systems were generated. These were then converted to LTS formulations, as needed by the CONNECTor Synthesis enablers. The Mealy machine and LTS formulations of the networked systems can be accessed at the dry-run website.
3. **CONNECTor Synthesis:** WP3 participants have undertaken the role of the synthesis enabler. Being part of the CONNECT process, they assumed to have as input: the LTS of each networked system, i.e. the ones of the German Consumer and of the French Merchant from the learning enabler, and the ontology mapping between their respective actions. Based on these inputs, they used the mediator theory, described in detail in deliverable D3.1, to synthesize the LTS of the CONNECTor that allows the consumer and merchant protocols to interoperate (available at the dry-run website).

The input and output files produced in the above process are accessible at http://www-roc.inria.fr/connect/connect-dry-run. Since our primary focus was to elicit the details of the CONNECTor *construction* process, the further activities of the WP5 related enablers (related to verification and validation) within the CONNECT architecture have not yet been integrated into the dry-run process at the time of writing of this deliverable. However, this work is ongoing and we envisage similar success. Note also that the online verification techniques developed by WP2 will be used to perform the Dependability Analysis activity.

5.1.3 Summary

This dry-run was a first extensive exercise in which multiple partners came together to address the issues involved in a concrete use-case of CONNECT. As far as giving the partners a better idea of the challenges which will be faced by the CONNECT system is concerned, it was a huge success. The interacting partners gained greater awareness into the requirements of their peers, and the effect they have on the working of their own enablers.

In summary, the lessons learnt from the popcorn dry-run were:

1. The output of the Learning enabler, the mealy machine, needs to be converted to LTS form for consumption by the Synthesis enabler. Although this is a straightforward process, it is important to note the difference.
2. A similar I/O mismatch exists for the inputs needed by the dependability enabler. We are working on exploring the details of this mismatch in the distributed marketplace context.

3. More thought needs to be given to the details of the ontology to be used by the CONNECT enablers.

We refer the reader to the deliverables D3.1 and D4.1 for the specific lessons learnt by each partner with respect to the operations of their individual enablers within the greater CONNECT process.

5.2 Implementing CONNECTors: a Bottom-up Approach to the CONNECT Architecture

5.2.1 Goals

The goal of this experiment is to develop a connected system from the bottom up, i.e., given two heterogeneous systems, implement a solution to connect them together. The purpose of this is to:

1. Create and deploy 'CONNECTors' in a running system.
2. Gain a greater understanding about the technical challenges that exist in attempting general interoperability solutions.
3. Identify how the key architectural elements of the CONNECT architecture will behave in a real system and more concretely what constraints real systems place on the CONNECT process.

Here a particular small scale scenario with a particular type of interoperability problem was first identified; subsequently an interoperability solution (specific to the scenario) was implemented directly. The scenario which we consider is that of a large gathering of people in a stadium, for example, for a concert or watching a football game. The scenario is partly inspired by one of the situations discussed in D 6.1 Section 2.5. In case of emergency, e.g, a fire or a structural lack of integrity, individuals should be able to get warning messages guiding them to proper exits. Sensors in exit tunnels can be used to even out the human traffic. We look at the flow of the application from the following perspectives.

- User: When a user enters the stadium, he uses his smart phone to discover the various warning services available, and subscribes to the ones he is interested in. At this phase, he also can choose to provide his location, in order to receive more customized alerts.
- System: The sensors in the stadium can act together to determine if there is danger (e.g., structural damage to a seating section, or fire or chemical hazards to entire stadium). Then, with or without mediation by security personnel, messages can be sent out to those who have subscribed to alerts.

Case Examples. There is an unlimited number of possibilities in terms of the systems employed to implement the stadium's warning system and the users' application. Hence, we restricted the problem to one stadium and two client systems:

- The stadium system application is implemented atop a JMS middleware instance and disseminates messages to a Java Message Service (JMS) broker. The technology employed is JBoss' implementation of JMS.
- Client 1 is a Java smart phone which subscribes to a JMS broker to receive messages about warnings; the client application is written using Sun's implementation of JMS.
- Client 2 is an Android phone, which receives asynchronous SOAP messages about warnings from a server that knows its endpoint (URL).

5.2.2 Methodology

To perform this experiment, a single developer took the 'role' of the CONNECT architecture. That is, the developer attempted to behave equivalently to the CONNECT process—i.e. replacing automated tasks with human/programmer equivalents:

1. Monitoring and learning of each system's behaviour. The developer used prior knowledge, system documentation and observation of the individual running systems to identify how each networked system behaved.
2. The synthesis of CONNECTors between two end-systems. Based upon observation of the systems as described previously, the developer designed, manually implemented and deployed the CONNECTors.

Monitoring and Learning of the Stadium Message Producer

JMS is a message-oriented middleware, whose overall behaviour is illustrated in Figure 5.2. A message producer creates a queue or topic on a JMS server (broker); a queue is for one-to-one messaging, whereas a topic is for publish-subscribe behaviour. The stadium application is the producer and employs a topic and sends warning messages (*m*); the mobile phone clients (consumers) subscribe to the topic and receive the messages as event notifications. Note, both parties discover the broker and/or topic using a Naming Service.

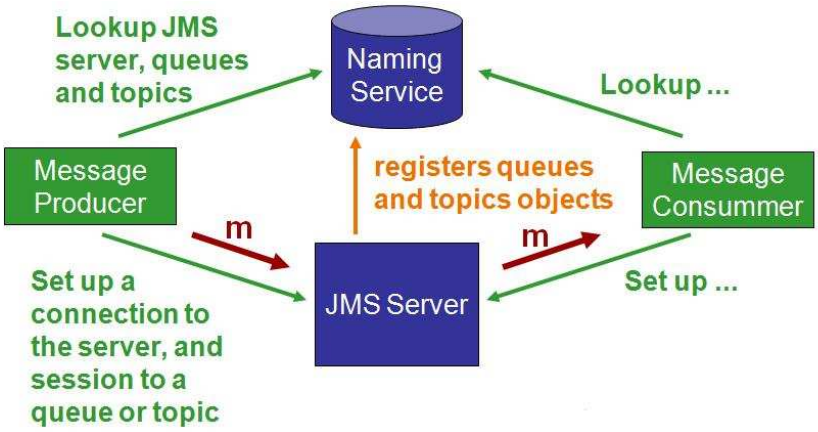


Figure 5.2: Overall architecture of the Java Messaging Service

An important feature of JMS is that it is an API standardized for portability reasons i.e. you can write a JMS application and it will execute on different vendor implementations of the JMS middleware. However, it does not standardize the exchange of messages; that is the messages that are sent using one vendor implementation of JMS are not understood by another vendor's implementations. Hence, the two cannot interoperate (even though they use the same middleware technology and architecture).

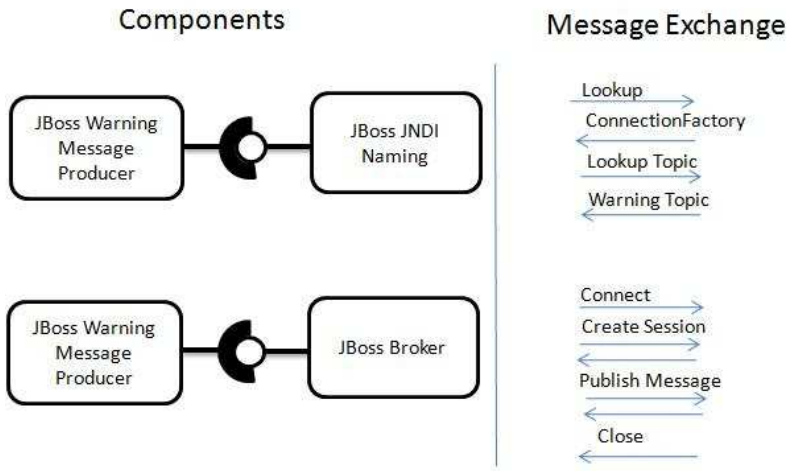


Figure 5.3: Behaviour of the Warning System Application

Through observation of the source code and runtime operation of the message producer stadium application implemented using the JBoss middleware—the sequence of operations between components in the distributed systems architecture was identified. The results of this observation are shown in Figure 5.3. Here the producer first uses the naming service to lookup a connection factory that it can use to create a connection with the broker, and then looks up the 'Warning' topic that it will send messages to. Subsequently, the producer communicates directly with the broker, first connecting and then creating a publication session. This session is then used to send a stream of warning messages. On completion the session and connection are closed.

Monitoring and Learning of Client One (The JMS client phone)

Through observation of the source code and runtime operation of the message consumer mobile client application implemented using Sun's JMS middleware-the sequence of operations between components in the distributed systems architecture was identified. The results of this observation are shown in Figure 5.4.

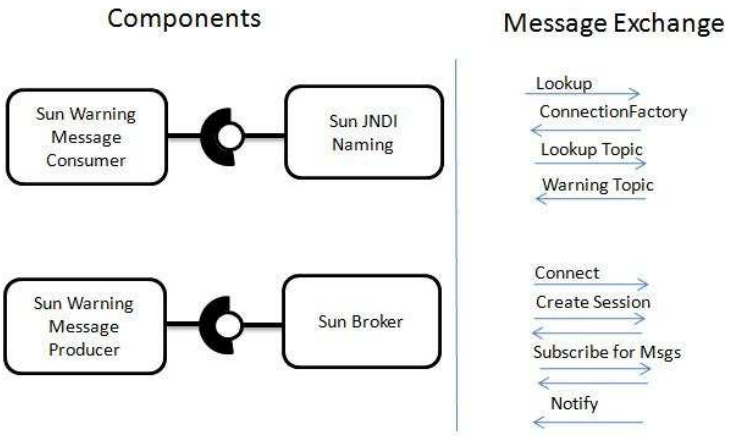


Figure 5.4: Behaviour of the Warning System JMS Client

The client application interacts with the naming service in a similar fashion to the producer and also creates a session on the discovered broker topic ('Warning'). Here it subscribes for messages, and whenever the producer sends a warning message it receives an event notification for this.

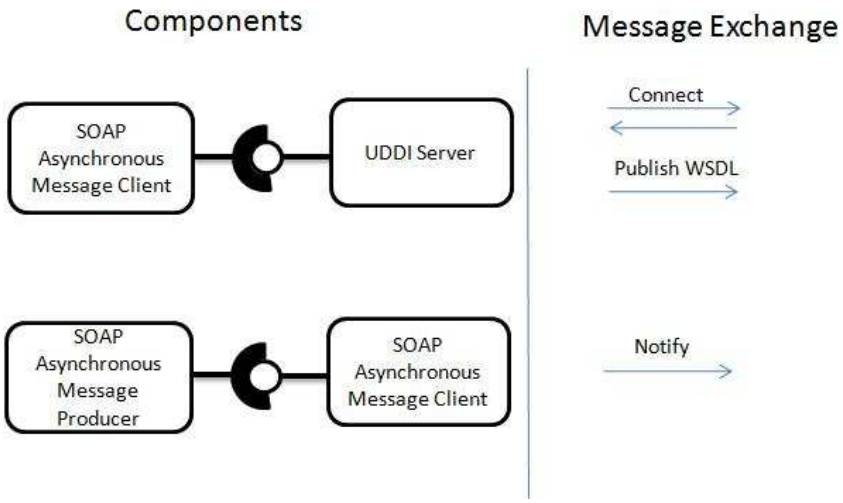


Figure 5.5: SOAP asynchronous messaging client behaviour

Monitoring and Learning of Client One (The SOAP client phone)

The SOAP client behaves very differently from the JMS solutions (as shown in Figure 5.5). First, a new client registers that it wants to receive warning messages by advertising its message endpoint; this is described by a WSDL file and then published to a UDDI Server (both of which are independent standards, like SOAP, and hence the vendor implementation of the middleware is unimportant). The SOAP producer (which is not deployed in the stadium) then searches for listening endpoints from the UDDI server and then sends a single asynchronous SOAP message to them directly.

Synthesis of Connector One: JMS Producer to JMS Consumer

JMS applications written upon different vendor implementations of the JMS API cannot interoperate. Hence, a CONNECTor is required with two different roles:

- A connection between the Sun message consumer and the JBoss JNDI Naming service (because a Sun JNDI implementation is not deployed)
- A connection between the Sun message consumer and the JBoss broker server.

The first connection was manually implemented as a single software component (deployed upon a third-party host). This component first allows the client consumer to connect to it and query it (as if it were a naming service)—the SUN naming service programming library was used to implement this functionality. All queries are intercepted and then passed onto the stadium's JBoss Naming service; this was implemented using the JBoss naming service programming library. *Note this approach relies on having the libraries of both middleware available, which has already been identified as a poor assumption in dynamic and pervasive environments.*

The implementation of the second connection follows a similar pattern; a dummy broker is implemented that first subscribes to the JBoss broker for the Warning topic (a simple subscription client implemented using the JBoss JMS API). The component is also implemented to allow the Sun client to connect to it (implemented using Sun's JMS library). An internal broker forwards messages from one 'side to the other' to ensure that the produced warning message reaches the client.

Connector Two: JMS Producer to SOAP Consumer

To create a CONNECTor between the JMS producer and the SOAP asynchronous messaging client a CONNECTor with the following two roles is needed:

- A component to receive UDDI publications and translate these into a subscription request for the JBoss broker.
- A connection between the producer to the SOAP client that translates JMS messages to SOAP messages.

The CONNECTor component was implemented as a UDDI listener that receives WSDL files (using an open source UDDI middleware). Upon reception of a WSDL file for a warning system listener endpoint, the component connects to the JBoss broker and sends a subscription request (this is implemented again as a simple JMS client using the JBoss programming library); this connects the stadium JBoss broker to the CONNECTor. When a warning message is produced, the CONNECTor component receives it and then the JMS to SOAP component translates the JMS text message into a SOAP message by embedding it within a SOAP envelope. Finally, the SOAP message is sent back to the consumer using a traditional http post of the SOAP message (implemented using an http programming library). The overall view of the implemented CONNECTor is shown in Figure 5.6.

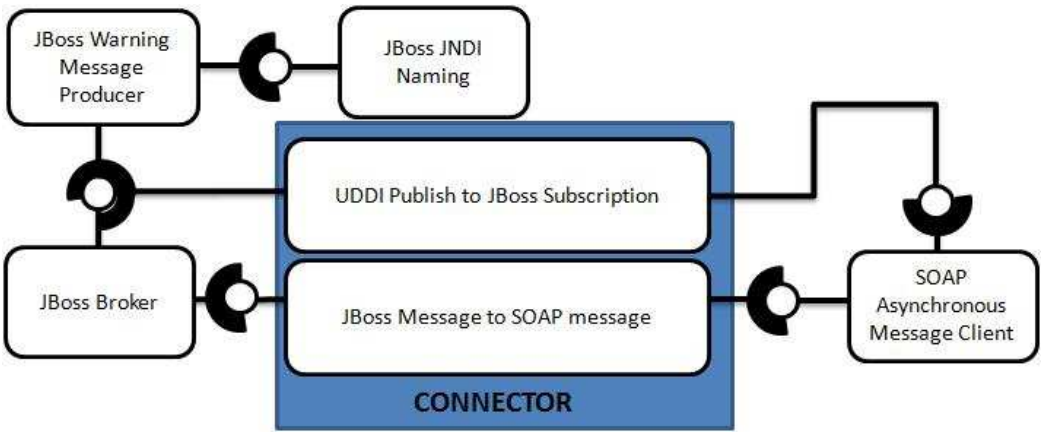


Figure 5.6: A CONNECTor between SOAP and JMS

5.2.3 Evaluation

With respect to the original aims of this experiment the following analysis can be drawn.

1. Create and deploy 'CONNECTors' in a running system.

The scenario shows that it is possible to connect highly heterogeneous systems with diverse communication paradigms e.g. JMS to SOAP; the SOAP application behaves inherently differently from JMS in terms of the way the discovery of other parties and communication proceeds. JMS employs a publish-subscribe abstraction, the SOAP client employs asynchronous messaging. Hence, these initial results that build CONNECTors to solve distinct interoperability issues across paradigm types are promising.

However, the deployment and implementation techniques employed to generate CONNECTors are standard with respect to previous approaches i.e. bridging methods. The implementation is static in terms of the two deployed systems and is implemented with design time knowledge of the systems and the programming libraries used to build them. Hence, the approach used to create the CONNECTors cannot be feasibly applied to spontaneous interactions in pervasive systems.

2. Gain a greater understanding about the technical challenges that exist in attempting general interoperability solutions.

The experiment uncovered two important technical challenges that may prove a hindrance to the generality of the CONNECT architecture and its ability to achieve universal interoperability. Hence, these are concerns to be addressed as the CONNECT architecture becomes mature.

- **Proprietary legacy middleware systems**. Observation of distributed systems and learning of their behaviour form a key part of the CONNECT procedure to resolve interoperability. However, when the application is developed atop a proprietary middleware (such as Sun JMS) information about message content, message format, data types, and other important CONNECTor information is not available at runtime (e.g. by observing communicated messages). This makes it difficult to learn a particular middleware's operation at runtime and also deduce how to connect to and interact with it. Hence, we will investigate rich self-description techniques and also new observation approaches to progress in this area.
- **Resources for CONNECT deployment.** The creation and deployment of CONNECTors is dependent upon resources in the network environment. First, there must be computational resources to deploy the CONNECTor (i.e. the software components must be loaded into memory and require CPU resources to execute); secondly, there must also be resources to execute the enablers (e.g. the synthesis of CONNECTors). These must be discoverable, and allow CONNECT software to run on them.

3. **Identify how the key architectural elements of the CONNECT architecture will behave in a real system and more concretely what constraints real systems place on the CONNECT process.**

Replacing the CONNECT enablers with a human programmer (as they have yet to be developed) provided a number of insights into how the CONNECT architecture would need to operate in practical terms. The different roles of the enablers are discussed in turn; however, the experiment was not complete in that it only considered some of the enablers and didn't implement monitoring and verification of operation, or attempt dependability analysis.

- **Discovery of Endpoint Systems** that are heterogeneous and require a CONNECTor to join them is the first step in the CONNECT process. The discovery enabler must respond to the discovery requests and behaviour of the legacy discovery protocols used to implement the system. The CONNECT discovery enabler must be aware of the different types of discovery protocols and the different ways in which systems advertise their services e.g. different naming services (JMS), different directories (UDDI), multicast protocols, etc. We believe that the discovery CONNECTors here (e.g. those that connected the client to the naming servers) will form part of the CONNECT discovery enabler.
- **Observation and Learning.** Experimentation with real systems has shown that learning and observation must take different forms. The discovery process can provide much of the starting information to begin to learn the system behaviour e.g. by discovering the interfaces of the JMS and SOAP protocols it is possible to learn the warning application behaviour. However, the observation of middleware protocols and learning of their behaviour presents many challenges: i) How to extract behaviour from observation of new packet formats? ii) How proprietary middleware that provides less information can be managed?
- **Synthesis of CONNECTors.** The experiment here used non optimal CONNECTors based upon pre-existing middleware libraries—this demonstrates the difficulties ahead for achieving synthesis of middleware protocols, especially where information about middleware behaviour is limited.

5.3 Resolving Data Mismatches

5.3.1 Goals

The goal of this experiment is to understand how we can exploit ontologies to resolve data mismatches across applications. Specifically, the objective is to push the envelope with respect to the ability to exploit the logic reasoning that is implicit in the ontologies to derive the mappings that solve data mismatches.

The scenario of the experiment is shown in the following figure 5.7 which shows two "smart" posters that have been augmented with Near Field Communication (NFC) tags. Specifically, the poster on the left shows four movies that are displayed at local theatres, as well as the theatres in which the movies run, the show times and the number of tickets that a mobile user may purchase. By touching the tags on the posters with an NFC enabled phone, a mobile user could buy a movie ticket to watch "Geisha" in the evening show. Similarly the mobile user may purchase transport tickets to move across the city. Clearly the posters are special user interfaces for services that sell either movie tickets or public transport tickets.

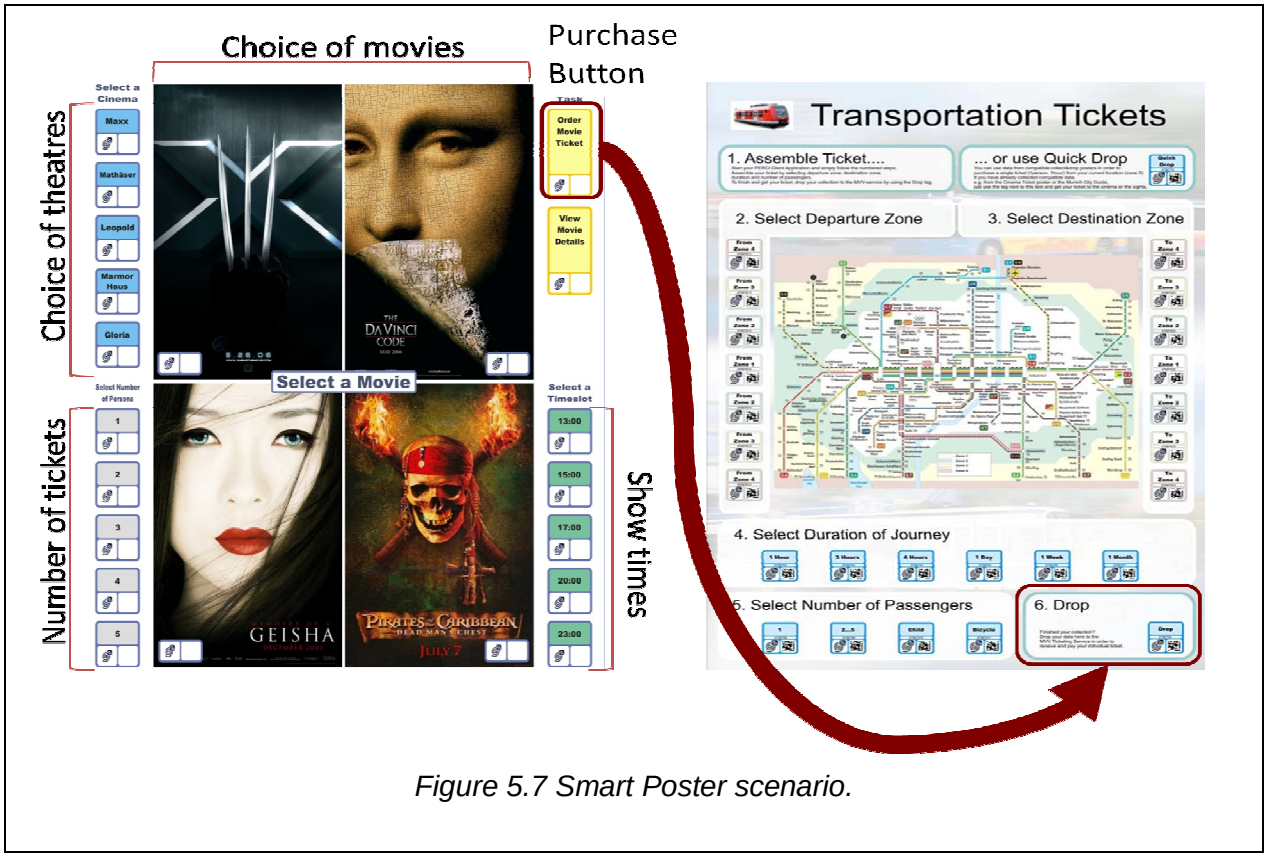


Figure 5.7 Smart Poster scenario.

Ideally, we would like to allow the mobile user to purchase the movie ticket, and then “drop” it in the transport poster to obtain the transport ticket. From the user prospective, the drop action is interpreted as a request to the transport poster to sell a ticket that takes the user to the movie. From the functional prospective, such a drop action is interpreted as a data passing from the movie ticket service to the transport ticket selling service.

One, somewhat obvious, requirement is that the two services are constructed independently, therefore the drop action cannot be specific for the movie poster, but it should rather be general enough to allow users to drop any data with an address within the scope of the Munich transport system. As a consequence, it would be unacceptable to rely on any transformation that is specialized for the movie poster.

The challenge of this experiment is therefore to produce a transformation of the movie ticket that automatically extracts the location of the movie theatre and passes it to the transport service. One complicating factor of such transformation is that the transport poster, which has been designed after the Munich ticketing machines, requires buyer to specify the zone in which the buyer would like to go, so the location extracted from the movie ticket should be also transformed in the corresponding zone.

5.3.2 Methodology

The two services are described in OWL-S and the mobile phone acquires the description of the services through the NFC interaction. Details on how such a process can be performed are described in the PERCI project [80]. What is relevant for us is that the two services map the data structures of the data that they exchange to different extensions of the same ontology.

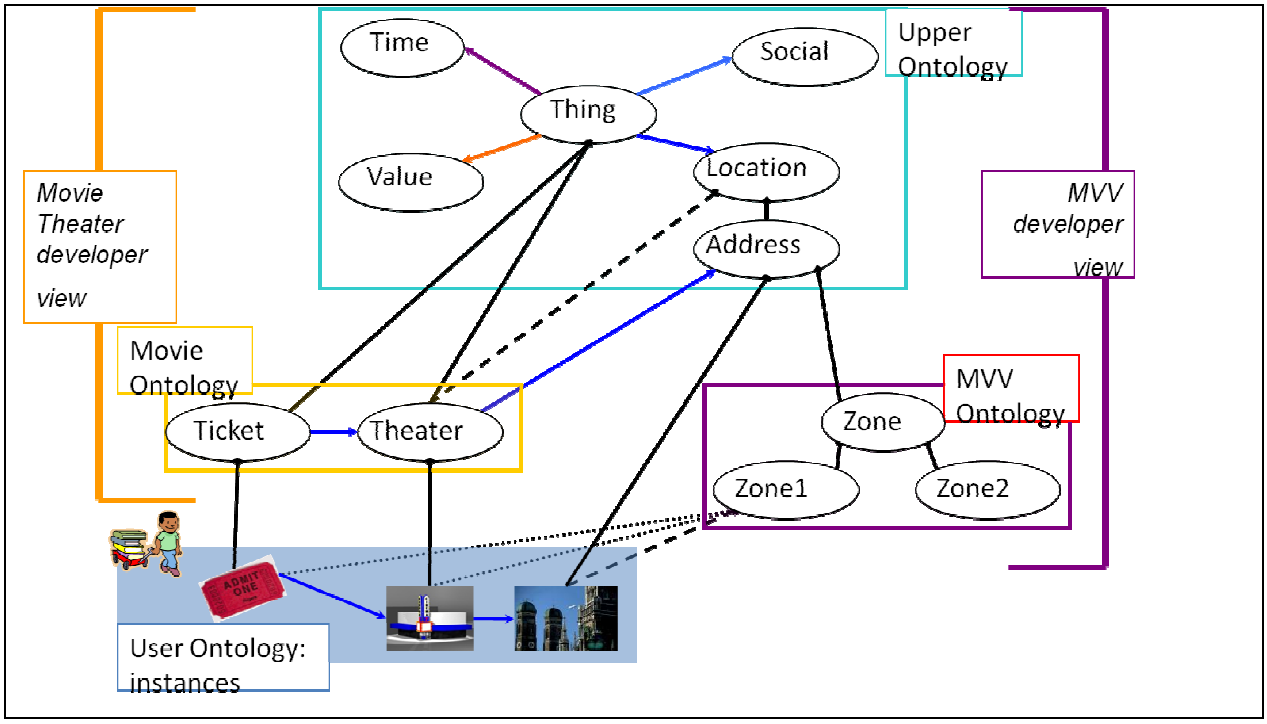


Figure 5.8 Ontologies in the Smart Poster scenario.

The structure of the ontology is shown in Figure 5.8. Specifically, there are three related ontologies that are used in this example. An "upper ontology" that provides generic definitions of concepts that may be relevant for different applications, in this case the concept `Thing` as the OWL most general class, and then four topic specializations: `Time`, `Value`, `Social` and `Location`. Furthermore, `Address` is considered a type of `Location`. The links between concepts represent different types of relations. Specifically, there are subclass relations, as well as relations between concepts. For example, a relation `atLocation(Thing,Location)` specifies the location where a given thing is, and in turn is used by the inference engine to classify the targets of the relation `atLocation` as `Locations` in the ontology.

The developers of the two applications provide further specializations of the upper ontology. The move service developer provides additional concepts such as `Ticket` and `Theatre`; while the MVV[5] service developer provides the concepts of zones which are defined as sets of addresses. For example, the concept `Zone1` is defined as all addresses whose city field is set to Munich. Note that since the relation `atLocation(Ticket,Theatre)` holds, `Theatre` is inferred as a subclass of `Location` by the inference engine (shown as a dashed line in the figure). Crucially, both the Movie Service developer and the Transport Service developer define their concepts as specializations of the shared Upper Ontology.

Upon interacting with the movie poster, the user's mobile phone would acquire a movie ticket, which is related to a given theatre, which in turn is an object whose location is a given address in Munich. When, at a later time, the user interacts with the transport poster, the mobile phone acquires also the MVV ontology. Since the address of the theatre is in Munich, it is classified as belonging to `Zone1` and therefore it is established a relation that goes from the ticket to the zone specification and such relation can be exploited to derive the input for the service.

[5] MVV stands for "Münchner Verkehrs- und Tarifverbund GmbH"; MVV is the Munich public transport authority

5.3.3 Evaluation

The example presented above shows how ontologies can be used to address the problem of data interoperability. Still a number of problems need to be addressed:

1. The derivation works with carefully constructed ontologies which have been constructed so that the derivation produces the correct results. A different issue would be to use more general and complex ontologies. To this extent, an interesting follow on experiment may be to use existing ontologies such as "Dolce"[6].
2. Whereas the use of ontologies above showed how the relation between the movie ticket and the zone can be established automatically, the exploitation of such a relation still requires some work. Specifically, it is quite easy to introduce axioms that force the classification of the movie ticket within the appropriate zone, but the adequacy of such axioms is quite dubious since they would force many objects to be also locations possibly conflicting with other quite natural negations.

5.4 Ontology Heterogeneity

5.4.1 Goals

The goal of this experiment was to survey, evaluate and select the best ontology matching tool for the CONNECT project. Ontology matching provides a possible answer to ontological heterogeneity problems [81] [82] [83] by identifying a set of relationships (i.e. equivalence (≡), subsumption (⊆), disjointness (⊥)) between ontological elements defined in two or more ontologies. The output of the matching process can be represented as a 4-tuple $<e_{iO1}, e_{jO2}$, rel., confidence> where the confidence value is a measure of the trust that the alignment is correct (typically in the interval [0,1]).

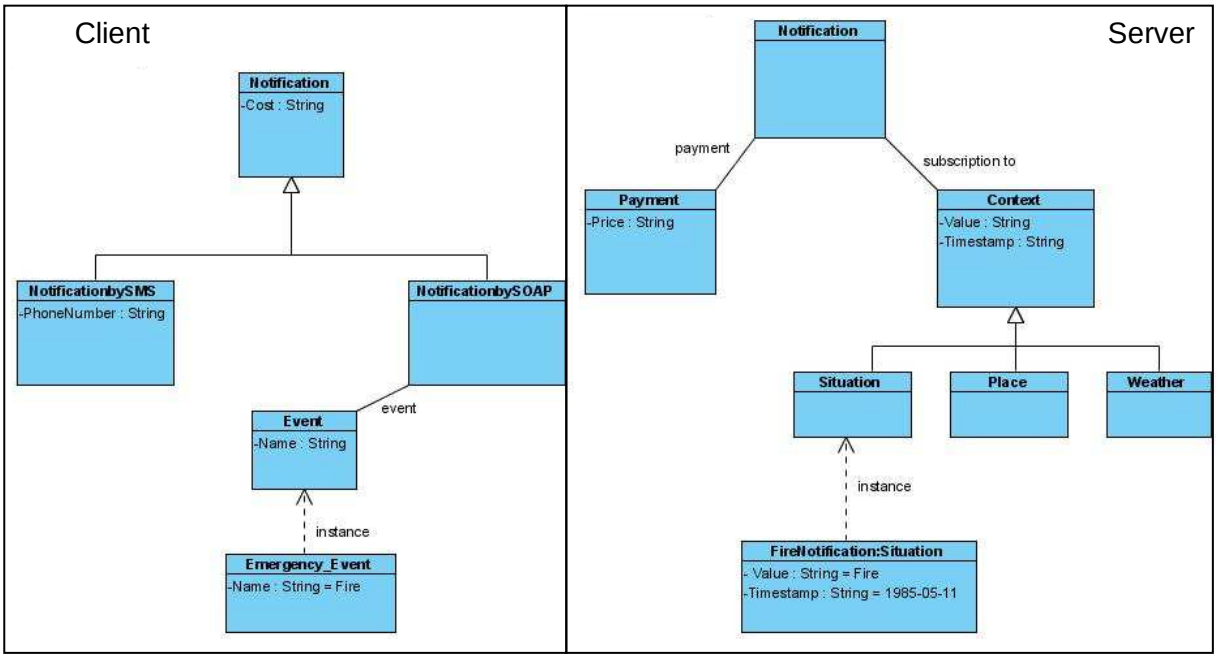


Figure 5.9: Notification Client Ontology (Left) and Notification Server Ontology (Right)

Let's consider the CONNECT Warning System scenario (as described in Deliverable D6.1 Section 2.4) as a motivating use case in which users can subscribe to notification services and get alert messages on their mobile phones when a danger occurs (e.g. fire in the stadium).

[6] http://www.loa-cnr.it/DOLCE.html

Let's further assume that ontologies have been developed independently for the mobile phone client and the notification server. As shown in the two UML diagrams (in Figure 5.9), while the client may use the parameter *Event* with a name (e.g. Fire) in the SOAP notification subscription request, the server ontology expects a Context or a more specific class (e.g. Situation) in the request.

Figure 5.10 shows the output of the ontology matching process based on the ontologies defined above. The ontologies are modelled in OWL[7], a knowledge representation formalism for the semantic Web.

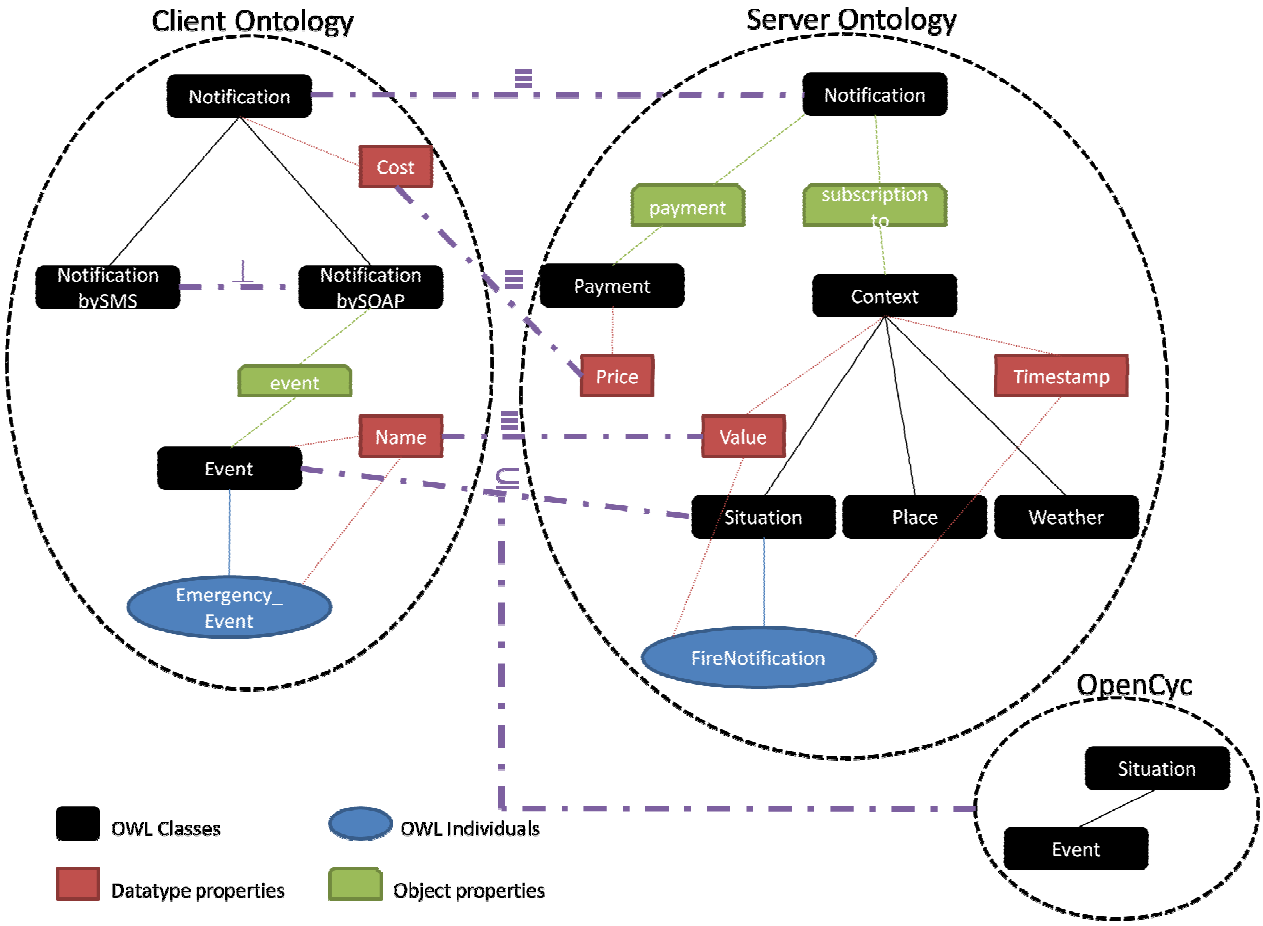


Figure 5.10: Ontology matching results

All equivalence relationships have been found automatically by the ontology matching tool OLA2. Besides, the subsumption relationship between the "Event" class and the "Situation" class has been manually added based on some background knowledge found in the upper ontology OpenCyc[8] that defines Event as a more specific class.

```
<Alignment>
<onto1>
  <Ontology rdf:about="http://connect-forever.eu/Notificationclient.owl">
   <location>**</location>
    <formalism>
     <Formalism align:name="OWL1.0"
      align:uri="http://www.w3.org/2002/07/owl#"/>
    </formalism>
```

[7] http://www.w3.org/TR/owl-ref/

[8] http://opencyc.org

```
  </Ontology>
 </onto1>
 <onto2>
  <Ontology rdf:about="http://connect-forever.eu/Notificationserver.owl">
   <location>**</location>
    <formalism>
     <Formalism align:name="OWL1.0"
      align:uri="http://www.w3.org/2002/07/owl#"/>
    </formalism>
  </Ontology>
 </onto2>
 <map>
  <Cell>
   <entity1
    rdf:resource='http://connect-forever.eu/Notificationclient.owl#Cost'/>
   <entity2
    rdf:resource='http://connect-forever.eu/Notificationserver.owl#Price'/>
   <relation>=</relation>
   <measure
    rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>0.63
   </measure>
  </Cell>
 </map>

 <map>
  <Cell>
   <entity1
    rdf:resource='http://connect-forever.eu/Notificationclient.owl#Name'/>
   <entity2
    rdf:resource='http://connect-forever.eu/Notificationserver.owl#Value'/>
   <relation>=</relation>
   <measure
    rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>0.64
   </measure>
  </Cell>
 </map>

</Alignment>
```

The SOAP notification request message can be now translated using the alignment results and further processed by the notification server (as illustrated in Figure 5.11).

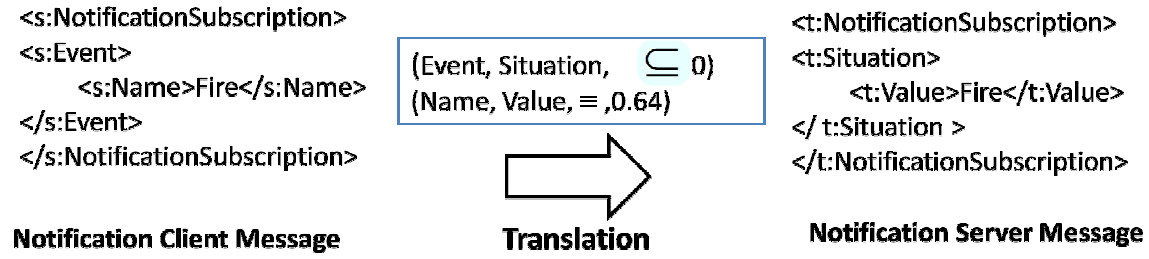


Figure 5.11: Translation of notification request message

5.4.2 Methodology

We restrict the study to the ontology matching tools that meet the following requirements:

- The tool shall align **automatically at run time** "semantically equivalent" elements which belong to two different ontologies. This is an essential requirement for CONNECT. Ontology

matchers aim at dropping one dimension of the heterogeneity barriers (i.e. ontological heterogeneity) and this task shall be ideally done without any user intervention and at run time.

- The tool shall support ontologies expressed in OWL. OWL is the de facto standard for semantic technologies and widely used to create ontologies on the Web.
- The tool shall have participated to at least one workshop on the evaluation of ontology matching (OAEI) which is organized at the yearly International Semantic Web conference and shall have performed well for the challenging Web directory track test case [84] [85] [86].
- The tool shall be **open source** (or at least shall contain executable binary code), currently under development, **generic** (i.e. not specialized in a specific ontology domain), and **not** tuned to a specific benchmark suite. The rationale behind this requirement is that we want to incorporate into the CONNECT prototype the best academic ontology matcher and have the possibility to modify the source code.

As shown in the table 5.1 below, among the best ontology matchers that participated to the evaluation at the International Semantic Web Conference, only five tools could be downloaded and evaluated. Moreover, only FALCON-AO, OLA2 and COMA++ succeeded to run our test cases. These three tools will be shortly presented in the following subsections.

Table 5.1: Results of Ontology Matchers Comparison

Tool	OAEI 2006		OAEI 2007		OAEI 2008		Availability	Open Source
	Participation	Case Rank[9]	Participation	Case Rank	Participation	Case Rank		
FALCON-AO	**Yes**	**1**	**Yes**	**3**	**No**		**Yes**	**Yes**
OLA2	**No**		**Yes**	**1**	**No**		**Yes**	**Yes**
COMA++	*Yes*	*3*	*No*		*No*		*Yes*	*No*
Lily/Lily v2	*No*		*Yes*	*4*	*Yes*	*3*	*Yes*	*No*
Prior+	*Yes*	*4*	*Yes*	*2*	*No*		*Yes*	*No*
CIDER	No		No		Yes	2	No	
DSSim	No		No		Yes	1	No	
HMatch2	Yes	6	No		No		No	
RiMOM	Yes	2	Yes	4	Yes	6	No	

FALCON-AO

FALCON-AO provides two linguistic matchers (V-Doc and I-sub), one structural matcher (GMO) and a hybrid matcher (PBM) [87] [88] [89] [90]. This tool is open source and all ontology matchers implement the match() interface and the alignment data format is OAEI-compliant[10]. The linguistic matchers V-Doc and I-sub use string distance metric techniques to measure the degree of similarity between two ontological elements based on the observation that the same elements are likely to be modelled using similar names (e.g. ShoppingCentre

[9] Ontology matchers are ranked based on their F-Measure value for the Web directory test case

[10] http://oaei.ontologymatching.org/2009

and Shopping_center). This observation is of course not always valid (e.g. terms may have multiple meanings) and V-Doc not only compares the labels associated to the elements but also the comments or annotations added by the expert to clarify the meaning of the ontological element. As reported in [88], V-Doc performs better than I-sub but actually fails to run our test cases (lack of memory problems). The structural matcher GMO uses V-Doc to find initial alignments and therefore cannot be evaluated. We selected PBM in our evaluation which uses I-sub as linguistic matcher and aims at handling efficiently large-scale ontologies [90].

OLA2

OLA2 uses linguistic (e.g. WordNet lookup) and structure-based techniques to compute the similarity between ontological elements [91] [92]. The matching process is performed over several iterations and the similarity values for ontological elements that belong to the same category (i.e. classes, object properties) are adjusted depending on the similarity values at the neighbor elements of a given entity and the process stops if the similarity values do not change more than a certain threshold (default value: 0.01). OLA2 is open source and all matching algorithms implement the Alignment interface defined in the org.semanticweb.owl.align package of the Alignment API [81]. The output of the alignment results can be written to files using various file formats: RDF, HTML, C-OWL [93], OAEI-compliant[11], etc.

COMA++

COMA++ is an ontology/schema matching tool that supports several input data formats (i.e. database and XML schema, OWL-DL) [94] [95] [96]. Users of the system are able to select and combine various match strategies over one to several matching iterations. They may also approve or disapprove obtained alignments between two matching iterations. Besides, the system can store in a relational database and reuse previously approved alignment results or auxiliary information provided by users such as a list of synonyms to find new alignments. For instance, given two match results, *match12: [Ont1i<-> Ont2i, Confidence=C12i%]* and *match23: [Ont2i<->Ont3i, Confidence=C23i%]*, the following match is derived (assuming transitive property): *match13: [Ont1i<->Ont3i, Confidence=(C12i%+C23i%)/2]*.

Table 5.2 summarizes the main findings of the comparative study of the tools.

Table 5.2: Comparative study of Ontology tools

Tool	***Falcon-AO***	***COMA++***	***OLA2***
Open source	YES	NO	YES
External resources	None	User-defined list of synonyms and acronyms	WordNet
Alignment format	OAEI-compliant	Proprietary	OAEI-compliant
Persistent Storage	NO	MySQL	NO
Composition of Matchers	(linguistic+structural only)	Manually over several iterations	(linguistic+structural only)
Logical correspondences	Equivalence	Equivalence	Equivalence

We consider in our own evaluation of the selected tools three ontologies in the geospatial domain:

[11] http://oaei.ontologymatching.org/2009

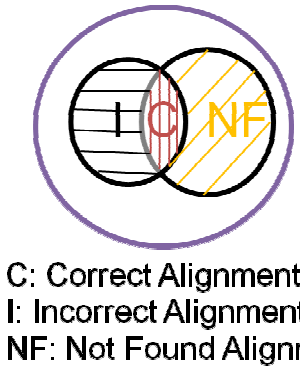
- IYOUIT Space Ontology[12] (IYOUIT SPACE)
- Ordnance Survey Building and Places Ontology[13] (OS BP)
- BOEMIE Geographic Information Ontology[14] (BOEMIE GIO)

These ontologies define spatial concepts and properties, partially overlap and are not known by the tools in advance. Each of them is shortly described in Table 5.3. The comparative evaluation is done by running blind tests in which the tools only rely on their algorithms to automatically match the ontologies. Furthermore, we perform a manual check of ontological alignments to get a better understanding of common mistakes and shortcomings of the current tools but also to compute the mapping precision for all use cases.

Table 5.3: Comparative study of geospatial ontologies

Ontology	Language	Number of Classes	Number of Datatype Properties	Number of Object Properties	Number of Annotation Properties
IYOUIT SPACE	OWL DL	167	4	20	3
OS BP	OWL DL[15]	686	0	25	14
BOEMIE GIO	OWL DL	290	29	52	2

The mapping precision is one performance indicator of the tools and is defined as the percentage of the correct alignments in all discovered alignments (illustrated in Figure 5.12). We have decided not to compute the mapping recall and as a consequence the F-Measure as this would require to identify by hand all correct matches (especially those not found by the tools).



$$Precision = \frac{C}{C \vee I}$$

$$Recall = \frac{C}{C \vee NF}$$

$$F\text{-}Measure = \frac{2.(Precision.Recall)}{(Precision + Recall)}$$

Figure 5.12: Quality indicators of ontology matching tools

[12] http://contextwatcher.docomolab-euro.com/Ontologies/1.1/space.owl

[13] http://www.ordnancesurvey.co.uk/oswebsite/ontology/v1.1/BuildingsandPlaces.owl

[14] http://mklab.iti.gr/project/boemieontology

[15] Original ontology language was OWL Full. All OWL individuals also used as OWL classes (OWL Full feature) have been removed

Figure 5.13 shows examples of valid alignments based on the Ordnance Survey and IYOUIT ontologies. The disjointness relationship between the class *RugbyStadium* and *Office* can for instance be explained looking at the definition of both classes in the ontologies:

Ontology O_1

a) RugbyStadium $\subseteq$ Stadium

Ontology O_2

b) Office $\subseteq$ Business_Place
c) Stadium $\subseteq$ Public_Place
d) Business_Place $\perp$ Public_Place

We also make a distinction between trivial and non-trivial alignments. Shopping_center and ShoppingCentre is one example of non-trivial alignment as this match would not be found by a naïve string comparison algorithm (e.g. Stadium ≡ Stadium is on the opposite trivial). Another example of a non-trivial case is to find the correct logical correspondence for FootballStadium and Stadium ($\subseteq$). A typical error would be to identify that FootballStadium and Stadium are equivalent classes.

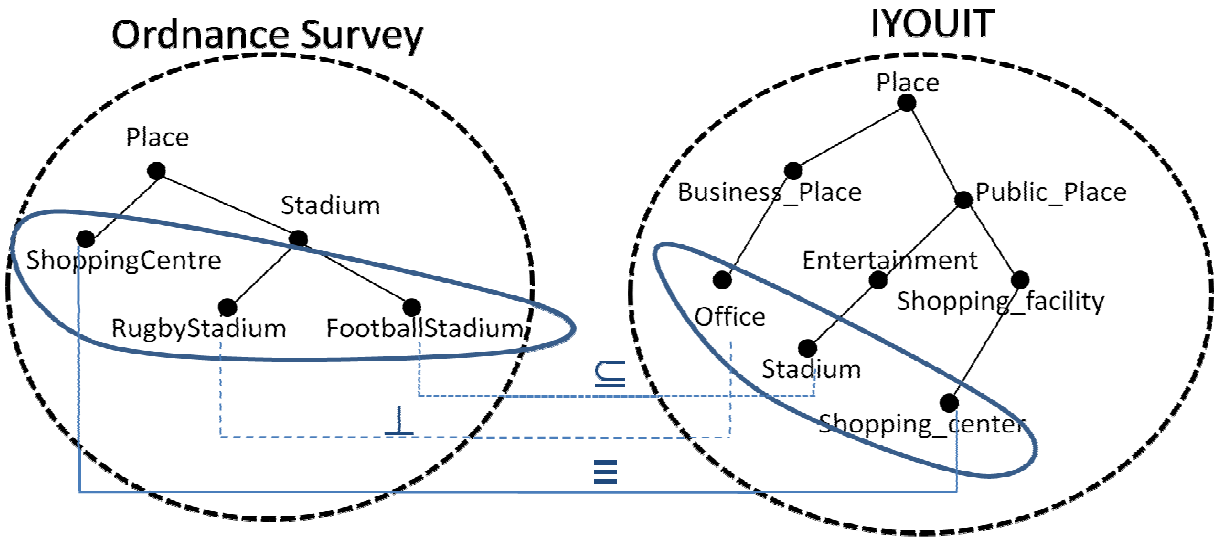


Figure 5.13: Examples of valid alignments

5.4.3 Evaluation

The main contribution of this initial experiment is to raise to CONNECT the ontological heterogeneity challenge which is particularly relevant for eternal systems where not only the data models may be developed and evolve independently between systems but also the semantic information attached to those systems.

We shortly discuss below the list of common mistakes and shortcomings of the evaluated tools The overall ontology tool evaluation results are illustrated in Table 5.4.

- Lexical confusions. The tools discovered incorrect alignments if both ontological elements were lexically similar (e.g. Port versus Sport). Moreover, the tools usually wrongly matched OWL classes named identically even though they were used in different contexts (e.g. Entertainment is defined as a Purpose versus Entertainment is defined as a Place).

- Lack of subsumption relationships support. All considered tools only provided equivalence relationships. As a consequence, tools were not able to discover subsumption relationships (e.g. Bowling_Club subsumed by Entertainment) or disjointness relationships (e.g. RugbyStadium $\perp$ Office). Besides, some of the discovered equivalence relationships shall have been replaced by subsumption relationships (e.g. FootballStadium and Stadium).
- Heterogeneous controversial mappings. Alignments were found between two ontological elements which belonged to different categories (e.g. matching of a OWL class to an object property).
- Incorrectness of the degree of confidence. We observed that an ontological alignment with a confidence of 1.0 may in certain cases be incorrect.
- Tuning of the confidence threshold. We modified the range of confidence values for FALCON-AO by adjusting the threshold parameter in the source code. As expected, the mapping precision was improved if the threshold was increased but at the same time, some of the non-trivial valid cases whose confidence value are below the threshold were filtered out (i.e. this actually means that the recall value deteriorates). The user intervention was still required to find the optimal threshold value that would ideally maximize the F-Measure value.

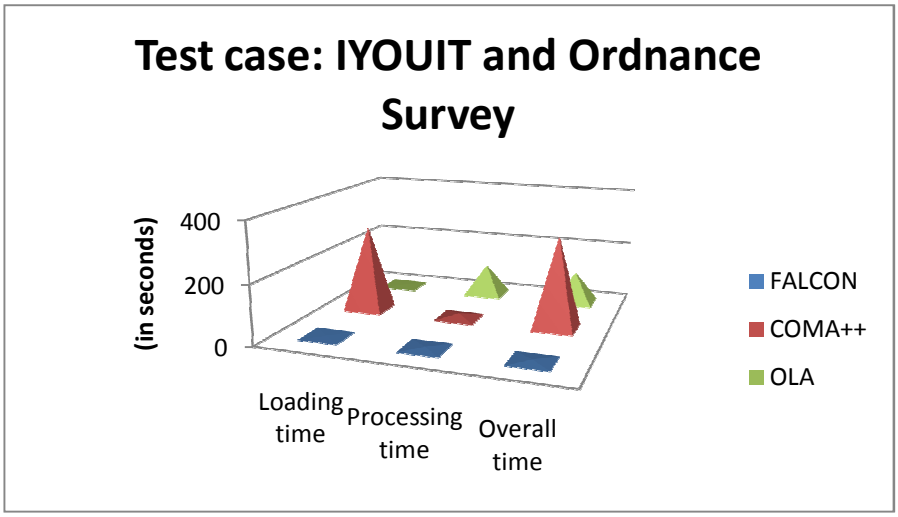


Figure 5.14: Time to perform matching for the three evaluated tools

Besides, a closer look at the evaluation results of last year's ISWC workshop reveal that for challenging real world use cases (e.g. Web directory or conference test cases [97]) the F-Measure value remains rather low (60%), few tools fulfilled our set of requirements and none of the available tools supported subsumption or disjointness correspondences.

The biggest challenge that we see is that a manual validation is still a mandatory step [98] and further research towards automatic ontology matching is needed [99]. Summarizing, as shown in the Table 5.4 and in Figure 5.14, FALCON-AO provides the best precision results for all the three cases and is much faster to compute the alignments. FALCON-AO, with respect to our knowledge, is the best open source tool for the CONNECT project.

Table 5.4: Overall ontology tool evaluation results

Test Case	Ontology Matcher	Number of alignments	Correct		Incorrect	Precision	Range of confidence values
			Trivial	Non-trivial			
IYOUIT SPACE & OS BP	FALCON-AO	78	44	10	24	69%	[0.75,1.0]
		66	44	10	12	82%	[0.8,1.0]
		62	44	10	8	87%	[0.85,1.0]
		56	44	8	4	93%	[0.9,1.0]
		55	44	8	3	95%	[0.95,1.0]
	COMA++	77	30	11	36	53 %	[0.4,0.7]
	OLA2	171	42	9	120	30%	[0.5,1.0]
IYOUIT SPACE & BOEMIE GIO	FALCON-AO	72	43	16	13	82%	[0.75,1.0]
		66	43	16	7	89%	[0.8,1.0]
		64	43	15	6	91%	[0.85,1.0]
		60	43	14	3	95%	[0.9,1.0]
		58	43	13	2	97%	[0.95,1.0]
	COMA++	77	42	21	14	82%	[0.4,0.7]
	OLA2	172	45	11	116	33%	[0.5,1.0]
OS BP & BOEMIE GIO	FALCON-AO	72	48	3	21	71%	[0.75,1.0]
		61	48	3	10	84%	[0.8,1.0]
		57	48	3	6	89%	[0.85,1.0]
		54	48	3	3	94%	[0.9,1.0]
		50	48	2	0	100%	[0.95,1.0]
	COMA++	70	29	13	28	60%	[0.4,0.7]
	OLA2	683	46	5	632	7%	[0.5,1.0]

5.5 CONNECTor Composition and Reuse

Generally speaking, the connection of two or more software peers can be realized in two ways, namely by the creation of a new *ad hoc* CONNECTor, or by composition of existing elements (i.e. components or CONNECTors). In Section 4, we discussed the interest of building and maintaining a knowledge base of connectors and of enabling connector compositionality and reuse. This section describes and discusses the pros and cons of the integration of the *composition approach* into the CONNECT architecture.

5.5.1 Overview

As explained, the runtime synthesis of software CONNECTors is a complex and resource-consuming task involving learning methods, analysis methods (data and protocol

interoperability solutions), code generation and so on. Performing such synthesis at runtime only makes sense if the reactivity requirement can be guaranteed.

Ensuring a sufficient reactivity can benefit from the storage and the reuse of existing CONNECTors, in order to connect (without new synthesis) similar peers discovered in the future. Figure 5.15 illustrates a possible CONNECT architecture leveraging the reuse of existing CONNECTors. When new peers require to be connected, the CONNECT enablers first look up into the repository to find suitable CONNECTors that have already been synthesized. They can search as well for possible assemblies that fit the needed connection. If nothing can be done by reuse, a new CONNECTor is synthesized and stored to populate the repository

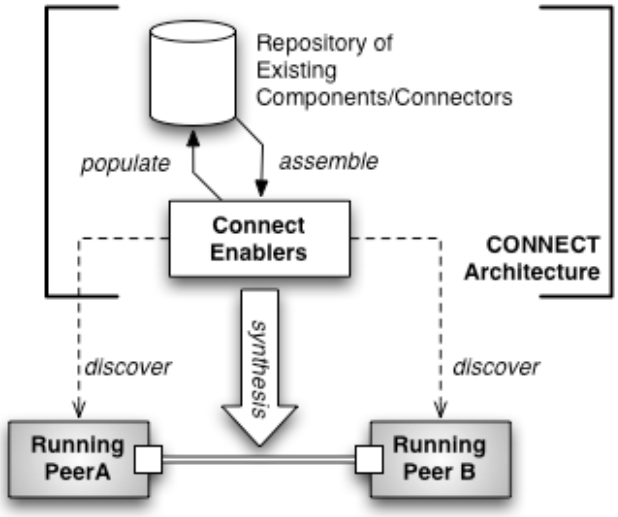


Figure 5.15 Enabling the synthesis of CONNECTors through components/CONNECTor composition

The experiment described below explores the use of such a repository of "already synthesized" CONNECTors in order to lighten the synthesis. It addresses three main challenges:

1. *Searching for existing* CONNECT*or solutions.* Finding a connection solution among existing CONNECTors is twofold. On one hand, a CONNECTor may have already been synthesized for the given connection need and it should therefore be reused. On the other hand, existing component/CONNECTors can the composed to build a CONNECTor without needing any new synthesis. Figure 5.16 illustrates the composition of two existing CONNECTors stored in the repository.

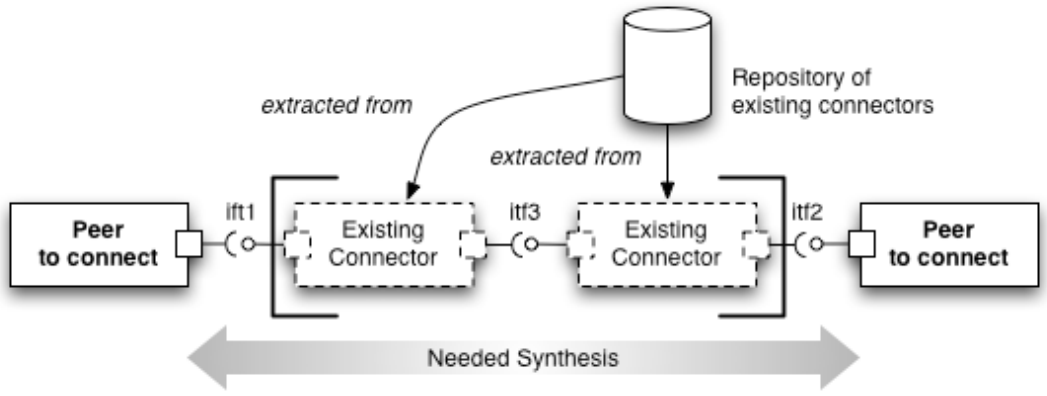


Figure 5.16: Challenge 1: Finding relevant assemblies of existing CONNECTors

2. *Selecting optimal existing solutions.* If several solutions can be built up from the repository, it becomes necessary to evaluate them in order to select the one that best fits connection requirements. Such a decision must takes into account non-functional properties such as performance, time-to-deploy, etc. Figure 5.17 illustrates the issues of the evaluation and the selection of optimal CONNECTor composition (w.r.t. given quality objectives).

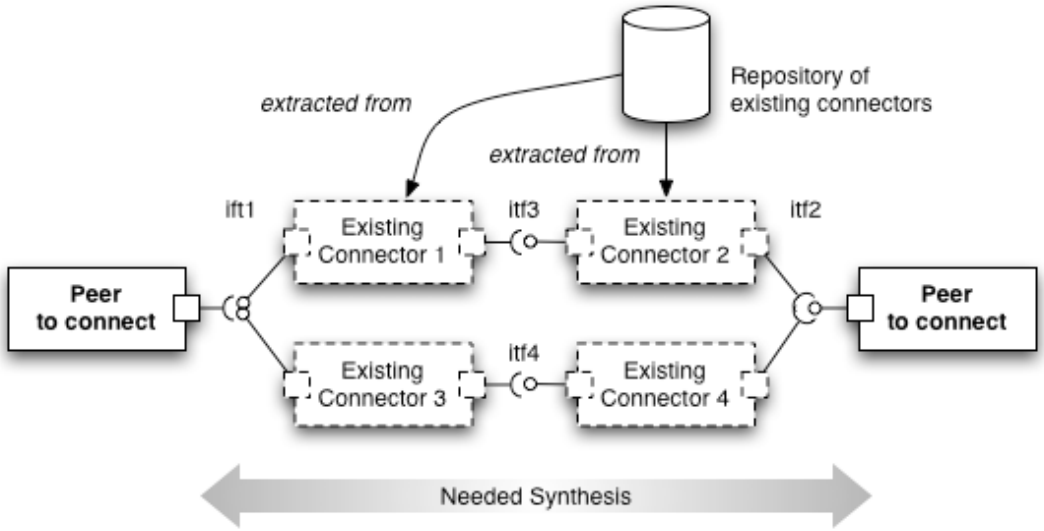


Figure 5.17: Challenge 2: Selecting the most relevant assembly

3. *Restricting the scope of the synthesis.* When the repository cannot provide any connection solution it may provide incomplete or partial solutions, which can be complete by a synthesis. In such a case, a trade-off is necessary between the complexity of the synthesis needed in both cases and the resulting quality in both cases. Figure 5.18 illustrates this last challenge: it shows that although no CONNECTor can be built from the repository, existing CONNECTors may lead to different synthesis needs.

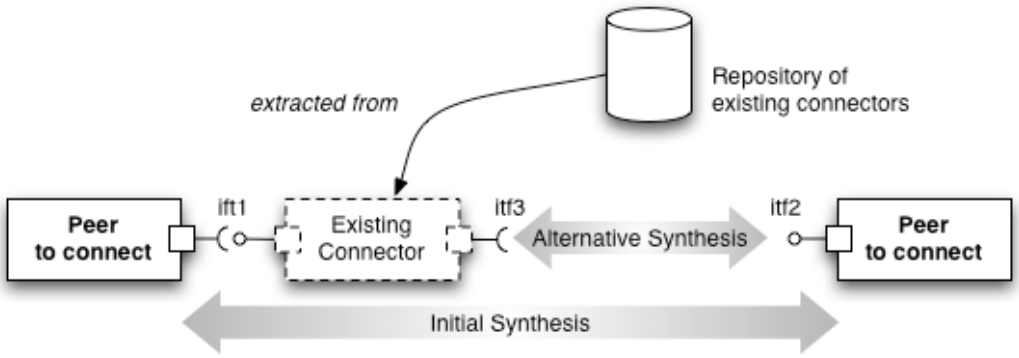


Figure 5.18: Challenge 3: Selection of most efficient Synthesis

5.5.2 Methodology

For the sake of simplicity, we abstracted the repository of existing CONNECTors as a set of UML component definitions (where the definitions are restricted to the structural elements such ports, interfaces, and operations). We extracted the needed subset of the UML component diagram related to our experiment.

***Searching for existing* CONNECT*or solutions*.** The repository can be organized as a directed graph where nodes represent CONNECTors and edges represent possible connections of CONNECTors. Figure 5.19 illustrates the organization of a repository containing CONNECTors depicted as UML components. Using such an organization, searching for composition of CONNECTors is boiled down to finding path into the graph of CONNECTors. In Figure 5.19, several connections between the peer A and the peer B are available in the repository since several paths CONNECT the related nodes.

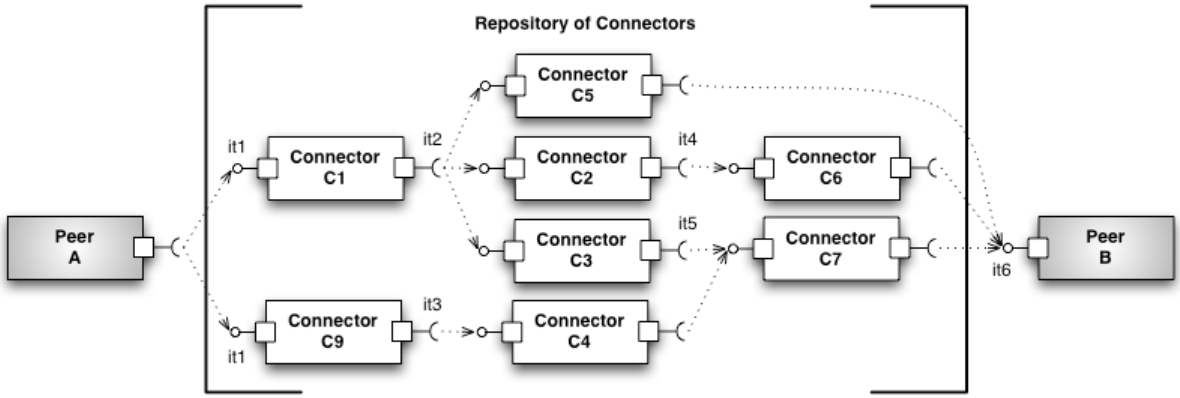


Figure 5.19: Organization of the repository of existing CONNECTors

Selecting optimal connection solutions. If several connections can be built up from the repository it's necessary to select the best one with respect to some quality objectives. Making such a decision implies to differentiate about the quality of the possible connections and thus requires some quality information about CONNECTors in the repository. We added quality information on interfaces, using parametric contracts [100]. In short, for a given component, parametric contracts define the quality of each provided service as a function of the quality provided at runtime to its required interfaces. Parametric contracts enable the end-to-end evaluation of quality on component assemblies and therefore on any connection built up from the repository. Selecting the best connection is finally done by the evaluation of each possible connection and by keeping the one that reflects the best trade-off with respect to quality requirements expressed by the two peers to connect.

5.5.3 Evaluation

The composition of existing CONNECTors in order to build new CONNECTors is an important tool within CONNECT'S arsenal as it has the potential to simplify the synthesis of complete CONNECTors and improve performance of the CONNECT process. This initial experiment demonstrates the feasibility of such techniques.

The effectiveness of composing existing CONNECTors (in contrast to synthesizing new ones from scratch) has not yet been evaluated, but it sounds reasonable to consider that reusing CONNECTors can save resources for simple cases. The synthesis of a new CONNECTor from scratch may result in good performance (in terms of the quality of connection); however, this may require runtime resources and take time to perform. In contrast, the composition of existing CONNECTors may require less resources and time but provide a lower quality connection.

The efficacy of such CONNECTor composition mining directly depends on the size of the repository and the complexity of the evaluation of the end-to-end quality of service of possible CONNECTors assemblies. However, the efficacy of the composition does not really matters here, because the complexity (and so the efficacy) of the whole synthesis process is obviously much larger.

Finally, although the challenge 3 of Section 5.5.1 has not really been investigated, it calls for the existence of methods to evaluate and anticipate the complexity of a given synthesis.

The runtime composition of existing CONNECTors also raises some technical issues, especially about connection solution deployment. If multiple instances of similar CONNECTors may be deployed in the same environment (for instance in a Fractal[16] architecture) the deployment is straightforward. However, such multiples instances are not allowed in some technological spaces (e.g. services oriented architecture) where deploying CONNECTor assemblies implies reusing existing and running instance of CONNECTors, and therefore implies that such CONNECTors support concurrent use.

In addition the quality model used to select the possible CONNECTors assemblies that best fit the quality required by the peers to connect may be difficult to get. A model such as parametric contracts is difficult to build; a design time but simpler model where the quality is expressed as crisp values could be learned or synthesized.

5.6 Overall Analysis

We took one final analysis step of the work produced in this report and considered as input: i) the general CONNECT architecture, and ii) the results and experiences gained from these experiments. Overall this identifies that we are making progress in the right direction in terms of meeting the original aim of the project i.e. making eternal network systems by bridging. However, this has also highlighted a number of additional challenges which should be considered. We now discuss these challenges in further detail.

5.6.1 Peer Discovery

Peer discovery is the starting point of the CONNECT process: CONNECT enablers must detect that peers exists, sense which of them must be connected with each other, and finally extract the relevant information required to synthesize CONNECTors.

Identifying peers that should to be connected. Once the CONNECT enablers are aware of the peers existing in the environment, it is then necessary to detect the ones which must be connected. This can be supported by a semantic matching on the intents inherent to each peers: if two intents match, then the two related peers must be connected.

Extracting data from the selected peers. Once two peers have been identified as potential partners, it is necessary to determine their technical characteristics i.e. in terms of system behaviour, architecture and protocols. Although the final knowledge driving the CONNECTor synthesis will be the result of the learning activity, the learning enabler itself requires some input information as well; such information must be discovered about or monitored directly on the peers.

Determining location of ontologies. CONNECTor synthesis requires ontologies; these will describe numerous elements to underpin the mappings between protocols and data respectively. However, such ontologies must be written by someone and discovered dynamically, or the ontologies must be learned from observation of the peers.

Scope and Applicability. The ability to bridge the gap between existing isolated network systems directly depends on the initial assumptions about peers/systems to connect: The stronger the assumptions, the less flexible the CONNECT architecture will be i.e. in all situations where that assumption does not hold. To get around this, the initial idea was to leverage existing learning techniques in order to dynamically discover existing peers and their characteristics, but such techniques have inherent assumptions as well. For instance, active learning techniques require that peers provide a testing API. The final scope and applicability

[16] http://fractal.ow2.org/

of the CONNECT architecture directly depends on the ability of the CONNECT project to minimize the number of such assumptions.

5.6.2 CONNECTor Synthesis and Code Generation

Synthesis of CONNECTors is a key element in resolving the interoperability problem, and for this process there remain important questions to answer.

Code Generation vs. Model Interpretation. The synthesis of CONNECTors is a two-fold operation. The inherent logic of the CONNECTor is synthesized as labelled transition systems that must still be converted into executable artefacts. Such a model transformation can result either in executable low level code such as Java or C++ or in executable models such BPEL. The first solution requires anticipation of the target platform on which the CONNECTor artefact is going to be deployed in order to use the relevant compiler and libraries. By contrast, the use of high level executable models enables a clear separation between code generation and deployment.

CONNECTor Deployment. Several deployment strategies for the synthesized CONNECTors can be planned: CONNECTors can be deployed either on third-party resources, such as a cloud or cluster computers, or directly on peers when no large-scale infrastructure is available. These two solutions make sense and will be further investigated.

Performance and Reactivity. Applying the CONNECT architecture to bridge between network devices such as mobiles phones raises the issue of connecting peers at the “network speed”. The topology of such networked systems is indeed likely to quickly evolve since devices may dynamically appear or disappear. Applying the CONNECT architecture therefore implies to fit this domain reactivity constraint both locally (for each tool/techniques involved in the tool chain) and globally (for the completed and integrated tools chain).

Dynamic Integration. The CONNECT architecture can be seen as a tool chain mainly involving discovery, learning, synthesis, etc. Although this three step principle remains generic, the running tool chain must be customized depending on the networked systems to connect. A technical solution is thus needed for the dynamic integration of the relevant tools into the chain. This may consequently contribute in addressing both the discovery and reactivity issues.

5.6.3 CONNECTor Life-Cycle

It is not sufficient to simply deploy CONNECTors; in order to achieve universal interoperability CONNECTors must be managed and react to changing conditions. This again poses important challenges to resolve.

Detection of Dependability Failures. Building eternally connected networked systems requires monitoring the behaviour of running CONNECTors in order to detect and react to faulty behaviour. Faulty functional behaviours may result from peer removals or peer failures whereas faulty non-functional behaviour may result from changing environmental conditions (e.g. hardware resources, etc). The detection of non-functional failures requires knowing the non-functional intent related to peers, which will be further investigated.

Restoring Dependability. Restoring dependability is basically a two-fold task including failure analysis and solution planning. Failure analysis aims at locating the reason of the failures (both for functional and non functional failures) whereas solution planning aims at inferring a change of the running CONNECTor which may restore the needed dependability level. Such changes may vary from a complete re-synthesis and re-deployment of the whole CONNECTor to a simple change of its configuration. The use and the feasibility of such changes must be further investigated.

6 Conclusions

The overall aim of the CONNECT project is to bridge the interoperability gap that results from the use of different data and protocols by the different entities involved in the software stack such as applications, middleware, platforms, etc. This aim is particularly targeted at heterogeneous, dynamic environments where systems must interact spontaneously i.e. they only discover each other at runtime. In this document, we have presented the initial vision of the CONNECT architecture that will meet this particular aim; this embraces and integrates recent advances and suggests the potential to revolutionize the state of the art in distributed systems and middleware.

In this report we first presented a detailed survey of the existing academic/research and industrial solutions to resolve middleware and data interoperability challenges. This survey represented interoperability solutions including i) traditional middleware such as CORBA or Web Services, ii) experimental technologies explicitly focusing on particular interoperability dimensions, and iii) semantics-based solutions to data interoperability. We identified a disconnection between the mainstream middleware work and the work on semantic interoperability; and also showed that none of the solutions address the requirements of for universal interoperability as identified in Section 2. Subsequently, we proposed the initial CONNECT architecture. This presented the key actors in a CONNECTed system i.e. networked systems, CONNECTors and enablers; and described the workflow operation of CONNECT to build and deploy a CONNECTor: from discovering peers, learning their critical characteristics and behavior, automatically synthesizing and verifying CONNECTors, to ensure that dependability requirements are maintained.

We provided a first a priori verification of our CONNECT solution as five experiments. The first one applies the CONNECT architecture in a top-down manner, investigating in particular how the integration of the work packages is achieved, and also how the partners technological expertise can be combined. By contrast, the second experiment started from concrete interoperability scenarios and operates in a bottom-up fashion in order to highlight real-world constraints limiting the applicability of the CONNECT architecture. In addition, we carried out two further experiments to anticipate potential technical challenges related to the realization of the solution. The third identified the role of ontologies within CONNECT to address the problems of data heterogeneity. The fourth reviews existing practical solutions to the data interoperability issue, and identifies one that can be integrated directly into the CONNECT architecture. The fifth extends our CONNECT solution and investigates possible solutions to leverage CONNECTor reuse and ensure performance requirements are met. From this work we identified a number of key challenges in the actual instantiation and deployment of the CONNECT architecture. This is particularly true of the CONNECT discovery enabler that will be a significant output of work package WP1; it is here that assumptions must be minimized in order for them not to restrict the flexibility of the CONNECT architecture. Further, integration of the chain of tools that combine for the CONNECT process remains the key-challenges towards an effective implementation of the CONNECT solution.

Overall, the lessons to be drawn from this document are i) in spite of the major research and industrial efforts to solve the problem of interoperability, current solutions demonstrably fail to meet the needs of modern distributed applications especially those that embrace dynamicity and high levels of heterogeneity, ii) the disconnect between middleware and semantic interoperability solutions severely hampers progress in this area, and iii) our concept of emergent middleware which intrinsically supports data interoperability and embraces learning, synthesis and verification techniques is a promising technique to address our requirements. The initial experiments have provided early evidence of the validity of the proposed approach and we look forward to a more refined understanding of emergent middleware over the remaining period of the project.

7 References

[1] Connect Consortium, "Emergent Connector for Eternal Software Intensive Networked Systems," FET Proactive 6: ICT Forever Yours Description of Work 2008.

[2] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Language and Systems*, vol. 7, no. 1, pp. 80-112, January 1985.

[3] M. Gudgin et al. (2007, April) W3C Recommendation. [Online]. http://www.w3.org/TR/soap/

[4] E. Guttman, C. Perkins, and J. Veizades. (1999, June) Service Location Protocol, Version 2. [Online]. http://tools.ietf.org/html/rfc2608

[5] Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright. (1999, October) Simple Service Discovery Protocol/1.0. [Online]. http://quimby.gnus.org/internet-drafts/draft-cai-ssdp-v1-03.txt

[6] UPnP Forum. (2008, October) UPnP Device Architecture 1.0. [Online]. http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf

[7] P. Bouquet, H. Stoermer, C. Niederee, and A. Mana, "Entity Name System: The Backbone of an Open and Scalable Web of Data," in *In Proceedings of the IEEE International Conference on Semantic Computing (ICSC 2008)*, 2008, pp. 554-561.

[8] M. Van Steen and A. Tanenbaum, *Distributed Systems: Principles and Paradigms*.: Prentice-Hall, 2001.

[9] Object Management Group, "The common object request broker: Architecture and specification Version 2.0," Technical Report 1995.

[10] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. (2001, March) Web Services Description Language (WSDL) 1.1. [Online]. http://www.w3.org/TR/wsdl

[11] D. Booth et al. (2004, February) W3C Working Group Note. [Online]. http://www.w3.org/TR/ws-arch/

[12] Microsoft Corporation. (2009, December) Distributed Component Object Model (DCOM) Remote Protocol Specification. [Online]. http://msdn.microsoft.com/en-gb/library/cc201989%28PROT.10%29.aspx

[13] R. Srinivasan. (1995, August) Network Working Group. [Online]. http://tools.ietf.org/html/rfc1831

[14] Microsoft Corporation. (2009) Microsoft Message Queuing. [Online]. http://www.microsoft.com/windowsserver2003/technologies/msmq/default.mspx

[15] Sun Microsystems. (2009) Java Message Service. [Online]. http://java.sun.com/products/jms/

[16] A. Carzaniga, D. Rosenblum, and A. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332-383, 2001.

[17] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford, "Tspaces," *IBM Systems Journal*, vol. 37, no. 3, pp. 454-474, 1998.

[18] J. Waldo, "Javaspaces specification 1.0," Sun Microsystems, Technical report 1998.

[19] N. Davies, A. Friday, S. Wade, and G. Blair, "L2imbo: A Distributed Systems Platform for Mobile Computing," *ACM Mobile Networks and Applications (MONET)*, vol. 3, no. 2, pp. 143-156, August 1998.

[20] A. Murphy, G. Picco, and G. Roman, "LIME: A Middleware for logical and Physical Mobility," in *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, Phoenix, AZ, 2001, pp. 524-533.

[21] OASIS. (2002) Univeral Description, Discovery and Integration of Web Services. [Online]. http://www.uddi.org

[22] M. Roman, F. Kon, and R. Campbell, "Reflective Middleware: From Your Desk to Your Hand," *IEEE Distributed Systems Online*, vol. 2, no. 5, August 2001.

[23] F. Kon et al., "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB, April 2000.," in *Proceedings of the 2nd International ACM/IFIP Middleware Conference,* New York, 2000.

[24] P. Grace, G. Blair, and S. Samuel, "A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, no. 1, pp. 2-14, January 2005.

[25] M. Duftler, N. Mukhi, S. Slominski, and S. Weerawarana, "Web Services Invocation Framework (WSIF)," in *Proceedings of OOPSLA 2001 Workshop on Object Oriented Web Services*, Tampa, Florida, 2001.

[26] S. Vinoski, "It's just a Mapping Problem," *IEEE Internet Computing Online*, vol. 7, no. 3, pp. 88-90, May/June 2003.

[27] Object Management Group, "COM/CORBA Interworking Specification Part A & B," 1997.

[28] Iona Technologies. (1999) OrbixCOMet. [Online]. http://www.iona.com/support/whitepapers/ocomet-wp.pdf,

[29] L. Brueckne. (2010, January) SOAP2CORBA. [Online]. http://soap2corba.sourceforge.net

[30] J. Miller and J. Mukerji, "Model Driven Architecture," Technical Standard OMG Document number ormsc/2001-07-01, 2001.

[31] P. Shah et al., "Interoperability between Mobile Distributed Components using the UniFrame Approach," in *Proceedings of the 41st Annual ACM Southeast Conference*, Savannah, Georgia, 2003, pp. 30-35.

[32] IONA. (2007) Artix ESB. [Online]. http://www.iona.com/products/artix/

[33] P. Raverdy, O. Riva, A. Chapelle, R. Chibout, and V. Issarny, "Efficient Context-aware Service Discovery in Multi-Protocol Pervasive Environments," in *Proceedings of the 7th International Conference on Mobile Data Management (MDM'06)*, Nara, Japan.

[34] Y. Bromberg and V. Issarny, "INDISS: Interoperable Discovery System for Networked Services," in *Proceedings of the IFIP/ACM/Usenix International Middleware Conference*, Grenoble, France, 2005, pp. 164-183.

[35] J. Nakazawa, H. Tokuda, W. Edwards, and U. Ramachandran, "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.

[36] N. Limam et al., "OSDA: Open service discovery architecture for efficient cross-domain service provisioning," *Computer Communications*, vol. 30, no. 3, pp. 546-563, 2007.

[37] C. Flores, G. Blair, and P. Grace, "An Adaptive Middleware to Overcome Service Discovery Heterogeneity in Mobile Ad Hoc Environments," *IEEE Distributed Systems Online*, July 2007.

[38] S. Zachariadis, C. Mascolo, and W. Emmerich, "Adaptable Mobile Applications: Exploiting Logical Mobility in Mobile Computing," in *Proceedings of 5th International Workshop on Mobile Agents for Telecommunication Applications", , October*, Marrakech, Morocco, 2003.

[39] K. Arnold, B. O'Sullivan, R. Scheifler, J. Waldo, and A. Wollrath, *The Jini Specification*.: Addison Wesley, 1999.

[40] J. Hammer and D. McLeod, "An approach to resolving semantic heterogenity in a federation of autonomous, heterogeneous database systems," *Int. J. Cooperative Inf. Syst.; 2(1)*, vol. 2, no. 1, pp. 51-83, 1993.

[41] M. Burstein et al., "DAML-S: Web Service Description for the Semantic Web," in *International Semantic Web Conference*, 2002, pp. 348-363.

[42] D. Martin et al., "Bringing Semantics to Web Services: The OWL-S Approach," in *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, 2004, pp. 26-42.

[43] David L. Martin et al., "Bringing Semantics to Web Services with OWL-S," *World Wide Web Journal*, pp. 243-277, 2007.

[44] J. Farrell and H. Lausen. (2007, August) W3C Recommendation. [Online]. http://www.w3.org/TR/sawsdl/

[45] D. Booth and K. Liu. (2007, June) Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. [Online]. http://www.w3.org/TR/wsdl20-primer/

[46] D. McGuinness and F. Harmelen. (2004, February) W3C recommendation. [Online]. http://www.w3.org/TR/owl-features/

[47] D. Martin, M. Paolucci, and M. Wagner, "Bringing Semantic Annotations to Web Services: OWL-S from the SAWSDL Perspective.," in *6th International and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, 2007, pp. 340-352.

[48] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, and L. Predoiu. (2005, October) D16.1v0.21 The Web Service Modeling Language WSML. [Online]. http://www.wsmo.org/TR/d16/d16.1/v0.21/

[49] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler, "WSMX - a semantic service-oriented architecture," in *Proceedings of the International Conference on Web Services (ICWS 2005)*, Orlando, Florida, 2005, pp. 321- 328.

[50] D. Jordan and J. Evdemon. (2007, April) Web Services Business Process Execution Language (WSBPEL) Version 2.0. [Online]. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[51] N. Kavantzas et al. (2005, November) Web Services Choreography Description Language Version 1.0. [Online]. http://www.w3.org/TR/ws-cdl-10/

[52] E. Cimpian, "WSMX Process Mediation," in *Second WSMO Implementation Workshop*, Innsbruck, Austria, 2005.

[53] E. Cimpian and A. Mocan, "WSMX Process Mediation Based on Choreographies," in *1st International Workshop on Web Service Choreography and Orchestration fo Business Process Management*, Nancy, France, 2005.

[54] Jos de Bruijn et al. (2005, October) http://www.wsmo.org/TR/d16/d16.1/v0.21/. [Online]. http://www.wsmo.org/TR/d16/d16.1/v0.21/

[55] S. McIlraith, T. Son, and H. Zeng, "Mobilizing the Semantic Web with DAML-enabled Web Services.," in *The Second International Workshop on the Semantic Web (SemWeb'2001)*, 2001.

[56] E. Sirin, B. Parsia, D Wu, J. Hendler, and S. NauDana, "HTN planning for Web Service composition using SHOP2," *Journal of Web Semantics 1(4)*, pp. 377-396, 2004.

[57] S. McIlraith and T. Son, "Adapting Golog for composition of semantic webservices," in *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR02)*, Toulouse,France, 2002, p. 482–493.

[58] S. Sohrabi, N. Prokoshyna, and S. McIlraith, "Web Service Composition via the Customization of Golog Programs with User Preferences," in *Conceptual Modeling: Foundations and Applications*. Berlin, Heidelberg: Lecture Notes In Computer Science, Springer-Verlag, 2009, pp. 319 - 334.

[59] P. Traverso and M. Pistore, "Automatic composition of semantic web services into executable processes.," in *Proceedings of the Third International Semantic Web Conference (ISWC2004)*, 2004.

[60] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities," in *International Semantic Web Conference*, 2002, pp. 333-347.

[61] N. Srinivasan, M. Paolucci, and K. P. Sycara, "An Efficient Algorithm for OWL-S Based Semantic Search in UDDI," in *SWSWPC 2004*, 2004, pp. 96-110.

[62] M. Klusch, Benedikt F., and K. P. Sycara, "OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 2, pp. 121-133, April 2009.

[63] M. Klusch, "Semantic Service Coordination. Semantic Service Discovery and Composition: A Survey," in *CASCOM - Intelligent Service Coordination in the Semantic Web.*: Birkhaeuser Verlag, Springer, 2008.

[64] R. Masuoka, B. Parsia, and Y. Labrou, "Task Computing – the Semantic Web Meets Pervasive Computing..," in *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, 2003.

[65] S. Singh, S. Puradkar, and Y. Lee, "Ubiquitous Computing: Connecting Pervasive Computing Through Semantic Web," *Information Systems and e-Business Management Journal*, 2005.

[66] D. Chakraborty, A. Joshi, and T. Finin, "Toward Distributed Service Discovery in Pervasive Computing Environments," *IEEE Transactions on Mobile Computing*, vol. 5, no. 2, p. 97–112, 2006.

[67] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, "Service Composition for Mobile Environments., 10(4):435–451.," *Journal on Mobile Networking and Applications, Special Issue on Mobile Services*, vol. 10, no. 4, pp. 435-451, 2005.

[68] S. Ben Mokhtar, N. Georgantas, and V. Issarny, "COCOA: COnversation-based Service Composition in PervAsive Computing Environments with QoS Support," *Journal of Systems and Software, Special Issue on ICPS'06*, vol. 80, no. 12, p. 1941–1955, 2007.

[69] S. Ben Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers, "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*, vol. 81, no. 5, pp. 785-808, 2008.

[70] M. Haas, E. T. Lin, and M. A. Roth, "Data integration through database federation," *IBM Systems Journal*, vol. 41, no. 4, pp. 578-596, October 2002.

[71] J. Jung, "Taxonomy alignment for interoperability between heterogeneous virtual organizations," *Expert Systems with Applications*, vol. 34, no. 4, p. 2721–2731, May 2008.

[72] J. Berlin and A. Motro, "Database schema matching using machine learning with feature selection," in *Lecture notes in computer science*.: Springer, 2002, pp. 452--466.

[73] J. Widom, "Trio: A System for Integrated Management of Data, Accuracy, and Lineage," in *Proc. of CIDR*, 2005.

[74] G. Vetere and M. Lenzerini, "Models for semantic interoperability in service-oriented architectures," *IBM Systems Journal 44(4)*, pp. 887-904, 2005.

[75] R. Fagin, P. Kolaitis, and L. Popa, "Data Exchange, Getting to the Core," in *Proceedings of ACM Symposium of Principles of Database Systems, ACM,* New York, 2003, p. 90–101.

[76] S. Narayanan and S. McIlraith, "Simulation, verification and automated composition of web services," in *WWW*, 2002, pp. 77-88.

[77] A. Ankolekar, M. Paolucci, and K. Sycara, "Towards a Formal Verification of OWL-S Process Models," in *International Semantic Web Conference*, Galway, Irland, 2005, pp. 37-51.

[78] Linking Open Data Wiki. [Online]. http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

[79] N. Mehta, N. Medvidovic, and S. Phadke, "Towards a taxonomy of software connectors," in *In Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, 2000, pp. 178-187.

[80] G. Broll et al., "Perci: Pervasive Service Interaction with the Internet of Things," *IEEE Internet Computing*, vol. 13, no. 6, pp. 74-81, November/December 2009.

[81] J. Euzenat and P. Shvaiko, *Ontology Matching*.: Springer, 2007, vol. ISBN 3-540-49611-4.

[82] P. Shvaiko and J. Euzenat, "Ten challenges for ontology matching," in *Proceedings of the 7th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE)*, 2008.

[83] S. Castano, A. Ferrara, A. Montanelli, G. Hess, and S. Bruno, "State of the Art on Ontology Coordination and Matching," BOEMIE Public Deliverable 4.4 2007.

[84] J. Euzenat et al., "Results of the Ontology Alignment Evaluation Initiative 2006," in *Proceedings of the 5th International Semantic Web Conference*, 2006.

[85] J. Euzenat et al., "Results of the Ontology Alignment Evaluation Initiative 2007," in *Proceedings of the 6th International Semantic Web Conference*, 2007.

[86] C. Caracciolo et al., "Results of the Ontology Alignment Evaluation Initiative 2008," in *Proceedings of the 7th International Smeantic Web Conference*, 2008.

[87] N. Jian, W. Hu, G. Cheng, and Y. Qu, "FalconAO: Aligning Ontologies with Falcon," in *Proceedings of the Integrating Ontologies Workshop*, 2005.

[88] Y. Qu, W. Hu, and G. Cheng, "Constructing virtual documents for ontology matching," in *Proceedings of the 15th International Conference on World Wide Web*, 2006.

[89] W. Hu and Y. Qu, "Falcon-AO: A practical ontology matching system," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 3, pp. 237-239, September 2008.

[90] W. Hu, Y. Qu, and G. Cheng, "Matching large ontologies: A divide-and-conquer approach," *Journal of Data and Knowledge Engineering*, vol. 67, no. 1, pp. 140-160, October 2008.

[91] J. Euzenat, P. Guegan, and P. Valtchev, "OLA in the OAEI 2005 alignment contest," in *Proceedings of the Integrating Ontologies Workshop*, 2005.

[92] J. Kengue, J. Euzenat, and P. Valtchev, "OLA in the OAEI 2007 evaluation contest," in *Proceedings of the 6th International Semantic Web Conference*, 2007.

[93] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt, "C-OWL: Contextualizing ontologies," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 4, pp. 325-343 , October 2004.

[94] H. Do and E. Rahm, "COMA – A system for flexible combination of schema matching approaches," in *Proceedings of the 28th International Conference on Very Large Databases*, 2002.

[95] D. Aumueller, H. Do, S. Massmann, and E. Rahm, "Schema and ontology matching with COMA," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2005.

[96] S. Massmann, D. Engmann, and E. Rahm, "COMA++: Results for the Ontology Alignment Contest," in *Proceedings of the 5th International Semantic Web Conference*, 2006.

[97] J. Euzenat et al., "Results of the Ontology Alignment Evaluation Initiative 2009," in *Proceedings of the 8th International Semantic Web Conference*, 2009.

[98] F. Shi, J. Li, J. Tang, G. Xie, and H. Li, "Actively learning ontology matching via user interaction," in *Proceedings of the International Semantic Web Conference*, 2009.

[99] A. Gal and P. Shvaiko, "Advances in Ontology Matching," in *Advances in Web Semantics I: Ontologies, Web Services and Applied Semantic Web*., 2008, vol. Lecture

Notes In Computer Science.

[100] V. Firus, S. Becker, and J. Happe, "Parametric Performance Contracts for QML-specified Software Components," in *Proceedings of the Second International Workshop on Formal Foundations of Embedded Software and Component-based Software Architectures*, vol. 141, 2005, pp. 73-90.