Reference Integration Architecture for Mobile Enterprise Solutions

Master’s Thesis

Carlos Andres Larrea Silva
Reference Integration Architecture for Mobile Enterprise Solutions

THESIS

submitted in partial fulfilment of
the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

Carlos Andres Larrea Silva
born in La Paz, Bolivia

Web Information Systems Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
http://eemcs.tudelft.nl

Working Tomorrow
Logica Netherlands B.V.
Rotterdam, the Netherlands
http://www.workingtomorrow.nu
Abstract

Nowadays, enterprises are facing several challenges associated with the design and implementation of mobile solutions for customers, employees and partners. One of the most important challenges is related to the integration between mobile applications and enterprise systems. In order to provide seamless integration, it is important to consider various aspects and challenges related to mobile solutions within the enterprise mobility strategy. This master thesis identifies and presents important challenges for enterprise mobility, mobile enterprise applications and mobile enterprise application integration, as well as various solutions and approaches to overcome them. Moreover, a reference architecture for integrating mobile enterprise applications and enterprise systems based on Web technologies is presented. The reference integration architecture describes the components, as well as the architectural and design patterns used to define its distribution, required to facilitate the integration between mobile enterprise applications and enterprise systems. Furthermore, the reference integration architecture can be used as a tool to identify infrastructure and integration requirements and to guide decision-making during the design and implementation of mobile enterprise solutions.

Thesis Committee:

Chair: Prof. dr. ir. G.J.P.M. Houben, Faculty EEMCS, TU Delft
University Supervisor: Dr. ir. A.J.H. Hidders, Faculty EEMCS, TU Delft
Committee Member: Dr. E. Visser, Faculty EEMCS, TU Delft
Committee Member: S. Lankester, Logica, George Hintzenweg 89, 3068AX, Rotterdam
Preface

This document has been produced within the master thesis project to obtain a master degree in Computer Science – Information Architecture at the Delft University of Technology. The project has been carried out within the Working Tomorrow programme and the Microsoft practice at Logica, and in cooperation with the EEMCS faculty of the Delft University of Technology.

This thesis marks the end of an important period in my life. Coming to the Netherlands has been a wonderful and enriching experience in many aspects. I had the opportunity to meet people from all around the world, learn about different cultures and make many new friends. Academically, I had the opportunity to learn and explore many new areas of scientific knowledge. Moreover, I have developed new skills and had the privilege of working with very talented and intelligent people from whom I have learned numerous things.

During this time, many people have contributed to my growth as a person and as a professional, and I am grateful to all of them. Let me begin by thanking God for all the blessings He has given me. I would also like to thank my parents, Mario Larrea and Maria del Pilar Silva, for their infinite love and encouragement, for being an example for me and for giving me the opportunity to study abroad. Likewise, I would like to express my eternal gratitude and love to my wonderful wife, Paola Tapia Pijuan, for all her love and unconditional support, and for being with me and making me so happy every day. People say that behind a great man is a great woman; well, I will strive to become a great man since I already have a great woman at my side. Moreover, I would like to thank my sister Claudia and my brother Roberto, for their love and support. Finally, I would like to thank my family and friends in Bolivia for always believe in me, and my friends in Delft for their support and all the amazing moments we have spent together during the last two years.

On the other hand, this thesis could not have been completed without the guidance, assistance and support of a number of people. I would like to thank my supervisor at the university, Jan Hidders, for his valuable comments and support during all the phases of my thesis and for his time and effort to evaluate my thesis report. I would also like to thank my mentor in Logica, Stefan Lankester, for his time, guidance and willingness to help me along the project. Similarly, I would like to thank Geert-Jan Houben and Eelco Visser, members of the graduation committee, for taking the time and effort to read and criticize my work. Moreover, I would like to thank my practice managers, Theo Joustra and Alex Lute, for their motivation and collaboration along the project. I would also like to thank Conny Svensson for his valuable insights and for sharing with me his experience in the field of enterprise mobility and Arie Leeuwesteijn for sharing with me his expertise in cloud computing and Microsoft technologies. Likewise, I would like to thank Hilke de Wit, Eric van Ommen, Gert-Jan Admiraal, Feitze de Vries, Barry van der Pol, Menno Bergeman, Nghia Le, Kevin Benedict and Andre Guillemin for their comments about my ideas and for guiding me in the right direction during the project. Finally, I would like express my gratitude to the Delft University of Technology and Logica for giving me the opportunity to work on such a challenging but exciting project.

I hope you enjoy reading this document as much as I did while working on it.

Carlos Andres Larrea Silva
Delft, the Netherlands
August 17, 2012
**Table of Contents**

Abstract ................................................................................................................................. i
Preface .................................................................................................................................... ii
Table of Contents .................................................................................................................. iii
List of Figures ......................................................................................................................... v
List of Tables .......................................................................................................................... vii

Part I: Background and Context .............................................................................................. 1

Chapter 1 Introduction .......................................................................................................... 2
  1.1 Research motivation ....................................................................................................... 2
  1.2 Research goal .................................................................................................................. 3
  1.3 Approach ....................................................................................................................... 4
  1.4 Structure of this document ............................................................................................. 5

Chapter 2 Enterprise Mobility ................................................................................................ 6
  2.1 Challenges of Enterprise Mobility .................................................................................. 6
  2.2 Challenges of Mobile Enterprise Applications .............................................................. 9
  2.3 Taxonomies of Mobile Enterprise Applications ............................................................ 12
  2.4 Mobile Application Architectures ................................................................................. 12
  2.5 Mobile Application Development Process .................................................................... 17
  2.6 Summary ....................................................................................................................... 18

Chapter 3 Mobile Enterprise Application Integration ............................................................ 20
  3.1 Challenges of enterprise application integration ........................................................... 21
  3.2 Challenges of mobile enterprise application integration .............................................. 21
  3.3 Enterprise Application Integration Approaches ........................................................... 28
  3.4 Web technologies and Web-based APIs ....................................................................... 33
  3.5 Summary ....................................................................................................................... 34

Part II: Reference Integration Architecture ............................................................................. 36

Chapter 4 Reference Integration Architecture Design .......................................................... 37
  4.1 Assumptions .................................................................................................................. 38
  4.2 Scenarios ....................................................................................................................... 39
  4.3 Requirements ............................................................................................................... 51
  4.4 Components and relationships ..................................................................................... 53
  4.5 Design decisions .......................................................................................................... 54
  4.6 Extension points ............................................................................................................ 64

Chapter 5 Integration Architecture Implementation ............................................................. 65
  5.1 Implementation scope and limitations .......................................................................... 65
  5.2 Frameworks and tools .................................................................................................. 66
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3 Implementation results and observations</td>
<td>77</td>
</tr>
<tr>
<td>Chapter 6 Integration Architecture Validation</td>
<td>79</td>
</tr>
<tr>
<td>6.1 Fictitious scenario</td>
<td>79</td>
</tr>
<tr>
<td>6.2 Frameworks and tools</td>
<td>80</td>
</tr>
<tr>
<td>6.3 Validation results and observations</td>
<td>85</td>
</tr>
<tr>
<td>Part III: Conclusions</td>
<td>88</td>
</tr>
<tr>
<td>Chapter 7 Conclusions and final remarks</td>
<td>89</td>
</tr>
<tr>
<td>7.1 Conclusions</td>
<td>89</td>
</tr>
<tr>
<td>7.2 Contribution of this work</td>
<td>90</td>
</tr>
<tr>
<td>7.3 Future work</td>
<td>90</td>
</tr>
<tr>
<td>References</td>
<td>92</td>
</tr>
<tr>
<td>Abbreviations</td>
<td>98</td>
</tr>
<tr>
<td>Glossary</td>
<td>99</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1 – Elements of a mobility-enabled architecture (Wu, et al., 2010) .................................................. 13
Figure 2 – Native mobile application architecture (XRG systems Inc, 2011) .................................................. 13
Figure 3 – Web-based mobile application architecture (XRG systems Inc, 2011) ........................................... 14
Figure 4 – Hybrid mobile application architecture (XRG systems Inc, 2011) .................................................. 15
Figure 5 – Mobile client approaches comparison (Worklight, 2011) ............................................................. 16
Figure 6 – Logica's approach for mobile solutions (Forslund, 2011) ........................................................... 17
Figure 7 – Taxonomy of enterprise application integration (Irani, et al., 2003) .................................................. 20
Figure 8 – Fuego implementation architecture (Tarkoma, et al., 2006) ........................................................... 24
Figure 9 – TANGO modular architecture (Wu, et al., 2010) ................................................................. 24
Figure 10 – The architecture of CAPNET middleware (Davidyuk, et al., 2004) ................................................. 26
Figure 11 – MobileSOA System Architecture (Ennai & Bose, 2008) .......................................................... 27
Figure 12 – Integration patterns and their relationships (Trowbridge, et al., 2004) .......................................... 28
Figure 13 – Classification of WOAs (Thies & Vossen, 2009) ..................................................................... 32
Figure 14 – Web-oriented architecture topology (Thies & Vossen, 2009) ....................................................... 33
Figure 15 – Components of the reference integration architecture .......................................................... 53
Figure 16 – Predefined service API styles in the reference integration architecture ..................................... 57
Figure 17 – Caching service across data centres from cloud provider ....................................................... 57
Figure 18 – Service Interceptor pattern with ingoing and outgoing pipelines .............................................. 58
Figure 19 – Overall design of services responsible for processing requests ................................................. 60
Figure 20 – Basic design for mobile clients ............................................................................................. 60
Figure 21 – Relation between service connectors and service providers ...................................................... 61
Figure 22 – Overall design of the Façade layer and its related components ................................................ 62
Figure 23 – Federated identity using a claim-based identity model. Adapted from (Chappell, 2011) ....... 63
Figure 24 – Elements considered for the implementation of the reference integration architecture .... 66
Figure 25 – Simplified version of the ASP.NET Web API stack diagram ....................................................... 68
Figure 26 – Example of a handler with logic for the ingoing and outgoing pipelines ................................ 69
Figure 27 – Adding message handlers to the HttpConfiguration .......................................................... 70
Figure 28 – Defining and registering routes ........................................................................................... 71
Figure 29 – Example of service controller implementation ........................................................................ 72
Figure 30 – Example of a method implementation within a service controller .......................................... 72
Figure 31 – Example of a Working role using a queue from the Service Bus Queue service .................... 74
Figure 32 – Windows Azure Service Bus Relay service. Adapted from (Skonnard & Brown, 2009) .... 74
Figure 33 – Authentication with ACS based on the claims-based identity model ..................................... 76
Figure 34 – Example of HTTP caching support within a method of a service controller ............... 76
Figure 35 – CachingHelper class using the Windows Azure Caching service.......................... 77
Figure 36 – CachingHandler using the Windows Azure Caching service. ............................ 77
Figure 37 – Overview of the implementation of the proof of concept................................. 80
Figure 38 – Internal logic architecture for enterprise data services.................................... 81
Figure 39 – Example of the interface for a data access component.................................... 81
Figure 40 – Example of the member of a domain component............................................ 82
Figure 41 – Example a service contract. ............................................................................. 82
Figure 42 – Resource model for the Web API. ..................................................................... 83
List of Tables

Table 1 – Assumption for the design of the reference integration architecture.............................. 38
Table 2 – Main concepts of the claim-based identity model (Baier, et al., 2011)............................. 63
Table 3 – Important classes within the ASP.NET Web API.............................................................. 67
Table 4 – Handlers implemented in the Façade layer. ..................................................................... 69
Table 5 – Elements of a route of the HttpRouteCollection. .......................................................... 70
Table 6 – Resources supported by service controller showed in Figure 29. ................................... 72
Table 7 – Summary of implemented requirements of the reference integration architecture............ 78
Table 8 – Enterprise data services used in the fictitious scenario. .................................................. 79
Table 9 – API structure for the Web API of the fictitious scenario. ................................................ 84
Table 10 – Supported HTTP verbs and media types for the Web API of the fictitious scenario....... 84
Table 11 – Service controllers for the Web API of the fictitious scenario...................................... 84
Part I: Background and Context
Chapter 1 Introduction

Enterprise mobility has been around for many years, but it has gained more attention during the last couple of years due to the revolution of mobile devices and mobile applications in the consumer market. Additionally, there have been significant improvements in the supporting mobile information and communications technologies in recent years. However, enterprise mobility is not limited to the use of mobile technologies in the enterprise context; it also allows people to access information and perform business processes independently from their location, resulting in more flexible enterprise organizations. The following definition of the mobile enterprise is based on three dimensions: 1) adaptability, 2) interaction and 3) access:

“The mobile enterprise is built on a foundation of processes and technologies allowing full access and instrumented insight to all organizational resources, resulting in improved adaptability, access, and interaction among employees, customers, partners, and suppliers, independent of location.” (Basole, 2007)

From the previous definition, it is possible to identify important aspects like the location independence, the diversity of users, the various supporting technologies required to enable it, and the impact of enterprise mobility at different levels of the enterprise organization.

Enterprise mobility can improve productivity by providing access to enterprise information anywhere, anytime, thus providing users with better decision-making support. Furthermore, enterprise mobility can facilitate the acceleration business processes and provide more flexibility to workflows, hence improving response time and reducing operational costs.

The benefits and drivers for enterprise mobility are clear. However, to successfully enable enterprise mobility, it is important to understand the related technical, organizational and economical aspects. This master thesis centres on some of the technological aspects of enterprise mobility, and in particular the technological aspects related to the integration between mobile enterprise applications and enterprise systems.

1.1 Research motivation

Due to the massive proliferation of mobile devices and mobile applications in the last years, people have experienced the benefits of having simple and easy-to-use applications that support many daily personal tasks and provide access to information anytime, anywhere. Now, people want a similar experience at work (Koushik, 2011).

In like manner, companies have realized the great benefits of mobility for their business and are starting to look at mobility from a more strategic perspective (Basole, 2007). Therefore, enterprises are trying to increase support for mobile solutions and provide more mobile applications to their employees, allowing them to be connected with the business anytime, anywhere through their mobile devices. The emergence of enterprise app stores and bring your own device (BYOD) initiatives are a clear proof of this consumerization of IT phenomena (Jones, 2011). With a proper implementation of an enterprise mobility strategy, enterprises can increase employee productivity, foster application innovation, improve mobility support, improve user experience with respect to enterprise applications, and promote business engagement.

However, enterprises face several challenges while implementing mobile solutions. One of the most important challenges is related to the integration between the mobile applications and the enterprise systems (Benedict, 2012). It is important to consider diverse aspects and various integration levels, as
proper integration can provide more flexibility to the business processes, more agility on the development of applications, and better the user experience, while continuing to be consistent with security and governance policies (Basole, 2007) (Marsh, 2011).

Consequently, to accomplish seamless integration, it is important to identify the current challenges in enterprise mobility, the current challenges related to mobile enterprise applications, the different types of mobile applications, and the common mobile application architectures. Furthermore, it is important to understand the current challenges of mobile enterprise application integration, identify the available supporting technologies and solutions, and define the infrastructure requirements to provide support for mobile enterprise application integration. All these aspects should be considered within an enterprise mobility strategy.

Currently, there are many mobile application development platforms and mobile middleware solutions that facilitate the development and integration of mobile solutions. However, the mobile market has proven to be very dynamic and change really fast, making difficult for enterprises to keep the pace. Moreover, enterprises have to integrate and manage mobile applications developed not only internally, but also by partners or external development teams. Finally, it is important to highlight the lack of portability between solutions from different vendors.

To sum up, there are different types of mobile applications, for different purposes and different target of users, and developed using frameworks and tools from different vendors. This diversity, along with the current heterogeneity of enterprise systems, presents a challenge for software architects and mobile development teams when designing and implementing mobile enterprise solutions.

1.2 Research goal

As a result of the above, this master thesis aims to determine how enterprises can use Web technologies to provide seamless integration between mobile enterprise applications and enterprise systems. Therefore, the main goal for this master thesis is:

Define a reference architecture based on Web technologies for integrating mobile enterprise applications and enterprise systems, allowing users to access enterprise information independently from their location and the type of mobile device or mobile operating system they use, while complying with security and governance policies.

The reference integration architecture may be used as a tool to identify enterprise requirements in terms of integration, and to guide decision-making during the design and implementation of mobile enterprise solutions. Enterprise architects and software architects can use the reference integration architecture as a baseline for the initial design of a particular mobile solution or to define an integration strategy as part of an enterprise mobility strategy. Therefore, the reference architecture can help identifying integration requirements, which may be used during the selection of mobile application development platforms or mobile middleware solutions.

The following research questions have been defined in order to accomplish the research goal:

- What are the current main challenges in enterprise mobility?
- What are the main characteristics and challenges related to mobile enterprise applications?
- What are the current types of mobile enterprise applications, and how they influence the mobile integration strategy?
What are the most common approaches for implementing mobile enterprise application used nowadays and what is their impact on the mobile integration strategy?

What are the current challenges for mobile enterprise application integration and what aspects, related to integration, have to be considered during the design and implementation of mobile enterprise solutions?

Which Web technologies can be used to enable and support integration between mobile enterprise applications and enterprise systems, and what are their infrastructure requirements?

The problem described above and these research questions are addressed in this master thesis.

1.3 Approach

To achieve the research goal presented above, the project has been divided in the following phases:

- **Research phase**: The research phase has been divided into two parts, a literature survey and a research within the mobile development teams in Logica. The purpose of the literature survey was to obtain more information about enterprise mobility, mobile enterprise applications and enterprise systems integration. The information obtained about enterprise mobility and mobile enterprise applications provides an overview and better understanding of the complexity of mobile enterprise solutions, the current state of this field, and allows to identify the infrastructure and technologies required to support this kind of solutions. On the other hand, the information obtained about enterprise systems integration facilitates the identification of approaches based on service-oriented architectures and Web technologies to overcome those challenges. Finally, the information obtained during the second part of the research, i.e. within the mobile development teams in Logica, provides a better understanding on the design and development process of mobile enterprise applications, the tools and technologies currently used, and the main integration challenges between these applications and enterprise systems. The research phase helped to:
  - Identify the current main challenges in enterprise mobility.
  - Identify the main characteristics and challenges related to mobile enterprise applications.
  - Identify the current types of mobile enterprise applications and their influence on the mobile integration strategy.
  - Identify the most common approaches for implementing mobile enterprise application used nowadays and their impact on the mobile integration strategy.
  - Identify current challenges for mobile enterprise application integration.
  - Identify which aspects, related to integration, have to be considered during the design and implementation of mobile enterprise solutions.
  - Identify the most common scenarios for integration between mobile enterprise applications and enterprise systems.

- **Analysis phase**: The information obtained in the research phase has been analyzed to derive the requirements and constraints for the design of the reference integration architecture.

- **Design phase**: Within the design phase, various iterations have been performed to satisfy the previously identified requirements, and several design decisions have been made and documented. In some cases, after a decision has been made, more research has been performed to achieve a more detailed design. For example, it was necessary to study in more detail the resource APIs as they have been used within the reference integration architecture.

- **Implementation phase**: For the implementation phase, an exploration of available technologies, frameworks and tools to implement the reference integration architecture has been carried out.

---


---

4
Then, the reference integration architecture design has been implemented using the selected technologies, frameworks and tools.

- **Validation phase:** For the validation phase, a proof of concept has been implemented, based on the fictitious scenario described in sub-section 6.1. Some enterprise data services have been developed to emulate the role of enterprise systems in a service-oriented architecture. Then, a Web API has been implemented to allow client applications access data and functionality exposed by the enterprise data services. Finally, some client applications have been developed to test the implemented architecture.

After each phase, an evaluation has been carried out to validate the results. The results of the research and analysis phases have been distributed to some people within Logica and TU Delft to receive comments and feedback. The results of the design have been shared with some experts within the mobile development teams in Logica, as well as with some external experts on enterprise mobility. In like manner, the implementation and proof of concept have been presented to the supervisors in Logica and TU Delft. Finally, some conclusions have been derived based on the results of all the previous phases.

### 1.4 Structure of this document

This master thesis document is divided into three parts. *Part I* comprises the background and context that provide an overview about the topic under discussion. *Chapter 1* presents the research motivation, the research goals, and the followed approach. *Chapter 2* presents the current state of enterprise mobility and mobile enterprise applications. It presents some challenges for enterprise mobility, some important challenges and considerations for the design and implementation of mobile enterprise applications, as well as some taxonomies and application architectures of mobile enterprise applications. *Chapter 3* presents the common challenges in enterprise application integration, the current challenges in mobile enterprise application integration, and some related work. Then it discusses some possible approaches for mobile enterprise application integration, and the use of Web technologies for this.

*Part II* is devoted to describe the proposed reference integration architecture for mobile enterprise solutions. *Chapter 4* explains the design of the proposed reference integration architecture. It starts by explaining the assumptions for the design, and describing the scenarios used during the design. Then it presents the requirements for the reference integration architecture, and explains the main design decisions. It concludes with a brief description of some extension points for the reference integration architecture. *Chapter 5* presents the scope for the implementation of the reference integration architecture and explains the limitations for that implementation. Then, it describes the frameworks and tools used for the implementation and ends with a summary of the implemented requirements. *Chapter 6* describes the fictitious scenario as well as the framework and tools used for the validation of the reference integration architecture implementation. It concludes with a detailed description of how a common request is processed within the implemented architecture.

Finally, *Part III* presents the conclusions of the master thesis. *Chapter 7* presents the conclusions and contribution of this master thesis. It concludes with some recommendations for future work.
Chapter 2  Enterprise Mobility

As mentioned in the previous chapter, enterprise mobility can bring a lot of benefits for businesses. Enterprise mobility can help businesses to achieve their goals by providing the ability to operate more effectively and to respond more quickly to changes through more flexible and agile processes (Redman, 2010) (Wu & Unhelkar, 2008). It can increase productivity, accelerate business processes, reduce costs, and improve the decision-making support (Basole, 2007) (Unhelkar & Murugesan, 2010). According to Gartner3 (Clark, 2011), mobile applications are becoming more influential in enterprises by improving efficiency and effectiveness of knowledge workers, by optimizing line-of-business processes, and by providing broader reach to customers and partners. Finally, according to a survey realized in the United States and Europe by Yankee Group3 (Marsh, 2011), the three main drivers for enterprise mobility are the improvement of responsiveness to customers, improvement of productivity, and improvement of operational efficiencies. However, besides all these benefits there are also many challenges related to enterprise mobility. Some of these challenges are related to privacy, security, management, and integration (Basole, 2007) (Sybase, 2011).

2.1 Challenges of Enterprise Mobility

2.1.1 Privacy

Privacy is a very important and delicate concern in an enterprise mobility strategy. Mobile devices are very personal since users carry with them all day long; therefore, allowing users to access enterprise applications from their mobile devices blurs the boundaries between personal and professional life as personal and enterprise data are accessible from the same device. This is true independently of who owns the device, the enterprise or the user (Koushik, 2011).

Consequently, enterprises need to keep personal data separated from enterprise data due to security or management reasons. This separation allows to establish stronger security and management policies over enterprise data without affecting user’s personal data, e.g. encrypting all enterprise data stored on the device, or being able to wipe all enterprise data if the device is lost or the person no longer works for the company. Of course this separation should be as transparent as possible, so the user experience is not compromised. Most of the current approaches are based on containers, i.e. the enterprise applications and data are placed within a container in the device. A disadvantage of this approach is the loss of application portability, because the applications need to be adapted to be executed within the container. Another approach is based on virtualization, i.e. adding a virtualization layer within the mobile device to support multiple instances running in parallel, e.g. one personal profile and a separate profile for work (Gasimov, et al., 2010). However, this approach is still immature.

2.1.2 Security

Another important challenge for enterprise mobility is security. According to the results of the survey realized in the United States and the United Kingdom by Kelton Research4 (Sybase, 2011), security is one of the main factors that prevent companies from implementing mobile solutions. Since security is a cross-cutting concern, aspects like confidentiality, integrity and privacy have to be considered at the client and server sides, as well as on the wireless networks used for communication between these two (Sathyan & Sadasivan, 2010). Inevitably, this increases the complexity of mobile solutions.

2 http://www.gartner.com/
3 http://www.yankeegroup.com/
4 http://keltonglobal.com/
2.1.2.1 Client-side security

At the client side, security has to consider aspects like malware, application/data provisioning and de-provisioning, access to local resources and services, and temporary local data storage (Mccammon, 2011) (Sybase, 2011). The main challenges are related to the resource limitations of mobile devices, the diversity of mobile devices and mobile operating systems (mobile OSs), and the protection of enterprise data outside the enterprise infrastructure boundaries. Usually enterprises have complete control over the IT infrastructure where enterprise applications run and enterprise data is stored; however, for mobile solutions this is not always the case, especially because nowadays the users are typically the owners of the mobile device. Because of this, security policies have to consider a variety of mobile devices and mobile OSs, as well as the user preferences. For example, it is not recommended to require a strong password to access the mobile device, because this can result in a bad user experience as the user has to introduce a long password even if he only wants to check the weather or simply make a phone call. Instead, data needs to be protected within the mobile device, e.g. encrypted. Additionally, it should be possible to remotely delete enterprise data from the mobile device. Many of these aspects are usually managed via mobile device management solutions (Redman, et al., 2011).

2.1.2.2 Server-side security

At the server side, the main challenges are associated with security integration, i.e. the need to extend the current security model to mobile solutions (Marsh, 2011). Here, security needs to deal with authentication and authorization to access enterprise applications. Enterprises want to control which devices and users can connect to the enterprise network, which applications are available to each user according to his role, and which mobile applications can access enterprise systems and enterprise information. Therefore, a high level of granularity is essential to satisfy these security requirements.

2.1.2.3 Wireless-network security

In wireless networks, security needs to deal with the same requirements and concerns than in fixed networks, i.e. confidentiality, privacy, integrity and availability, and some additional challenges due to the limitation of resources at the client side, the dynamism of the environment, and the broadcast nature of wireless communications (Tarkoma, 2009). Confidentiality guarantees that the information in the message is only accessible by authorized entities, and it is usually achieved through encryption at the message level. Integrity guarantees that the message is not modified during transmission, and that the involved entities are properly identified and authorized. The former is commonly achieved by securing the connection, e.g. using the Secure Sockets Layer (SSL) protocol, which can prevent security threats like man in the middle attacks. The latter is achieved through authentication and authorization mechanisms between the involved entities that permit to identify each other and determine what they are allowed to do. The main risks are related to spoofing attacks and identity theft. Finally, availability ensures that the service is constantly available for users, e.g. preventing denial of service attacks. For this, it is required to have support from the whole infrastructure, e.g. storage, processing, communication.

2.1.3 Management

Another important challenge for enterprise mobility is related to the management of mobile solutions. As discussed before, the implementation of enterprise mobility has consequences not only at the technological level, but at the organizational and economical levels as well. Therefore, enterprises need to determine beforehand how to manage different aspects related to mobile solutions on each of these levels.
Governance has an important role in the implementation of mobile solutions. Enterprises need to extend or create governance mechanisms to manage many aspects related to enterprise mobility (Koushik, 2011) (Redman, 2010). Traditional enterprise solutions are designed for highly controlled and monitored computing environments with defined computing capabilities and standardized clients, but this is usually not the case for mobile enterprise solutions. Consequently, policies and processes become more complex. For example, development, testing, deployment and supporting processes related to mobile solutions are more complex due to the diversity of mobile clients. Additionally, it is important that governance mechanisms define the required roles to support these policies and processes, as well as the decision-making responsibilities for each of these roles (Peterson, 2004).

Governance has to consider many aspects related to mobile solutions like application development, application distribution, user management, risk and security management, and expenses management. Thus, enterprises need to define specific policies and processes to effectively manage these aspects. All this has impact in the process and the integration strategy. For example, if a company decides to implement a BYOD programme, then it has to consider the impact on some processes, e.g. the mobile application development process, as cross-platform support may be needed (Jones & Wallin, 2011). Therefore, it is important to define what new skills and capabilities are required to support these processes, as well as to implement and monitor the policies.

Nowadays, there are two main approaches for the management of mobile enterprise solutions, viz. a device-centric approach and an application-centric approach. The device-centric approach is focused on device management. Currently, there are several Mobile Device Management (MDM) solutions in the market, and it is common that these solutions provide support for multiple mobile devices and multiple mobile OSs (Jones, 2011). On the other hand, the application-centric approach is focused on application management. As the number of mobile applications grows in an enterprise, there is a need to manage the lifecycle of these applications and to provide new ways for application distribution. Therefore, Mobile Application Management (MAM) solutions have emerged to satisfy these needs. The selection of one of these approaches or a combination of both depends on the enterprise mobility strategy.

2.1.4 Integration

Finally, but not less important, enterprise mobility brings challenges in terms of integration. Integration has to be achieved at different levels on the organization and from different perspectives. Enterprise mobility requires integration at the various levels of the enterprise architecture, i.e. business processes, information, application, technological. Additionally, it is not enough to consider integration from a technological perspective; it has to be seen from a broader perspective, including organizational and economical aspects.

From the technical perspective, enterprise mobility integration has to consider aspects like data integration and optimization, application integration, and security integration among others (McKeen & Smith, 2002). Usually many of these aspects are supported by middleware services. However, each enterprise has a unique landscape of enterprise applications, with different technologies, challenges and objectives. Therefore, a middleware solution or a mobile application platform solution is not always enough. Enterprises need to extend their integration architectures and integration strategies to provide seamless integration with mobile applications.

In like manner, enterprise mobility integration has to consider aspects related to business processes. Enterprise mobility should provide more flexibility for business processes, allowing users to execute some business tasks from their mobile devices at any time, and independently from their location.
With such an event-oriented integration approach, the general process speeds up as there is no need to wait until the employee returns to his desk to execute the task, thus giving more flexibility and agility to the process. Again, for this to happen, seamless integration between mobile applications and enterprise systems is required.

2.2 Challenges of Mobile Enterprise Applications

After discussing the challenges in enterprise mobility, it is time to take a look at the challenges in mobile enterprise applications. Mobile enterprise applications have some similarities with traditional enterprise applications, but they also have unique requirements and concerns. Mobile enterprise applications extend the challenges of traditional enterprise applications. Therefore, different aspects related to the mobile environment and enterprise environment have to be considered during the design and implementation of these applications.

2.2.1 Mobile environment considerations

Mobile computing systems are designed for dynamic environments; therefore they usually have the following capabilities and characteristics: wireless network connectivity, context-awareness, limited power supply, and limited device capabilities e.g. processing power and storage. Moreover, nowadays there is a variety of mobile OSs and the market is constantly evolving, which brings the need for cross-platform support. Therefore, common challenges and issues associated to mobile applications are related with adaptability, resource constraints, and the diversity of mobile devices and mobile OSs (Ennai & Bose, 2008).

2.2.1.1 Adaptability

Due to the dynamism of the mobile environment, mobile applications need to be able to constantly adapt in response to changes in the context. Therefore, mobile applications need context-awareness support, i.e. information and the application functionality has to be adapted according to these changes in the context where the application executes. However, providing context-awareness support in mobile applications is challenging (Capra, et al., 2003) (Issarny, et al., 2007). Some strategies for context-awareness and adaptation support are presented in (Satyanarayanan, 1996). Nevertheless, there are different context dimensions that should be considered in addition to an adaptation strategy, i.e. the type of context information used in the adaptation process, e.g. system context, related to availability of systems resources, and user context, related to parameters like location, time and user profile (Kakousis, et al., 2010).

Mobile applications need to use services from the mobile device to collect this context information, e.g. location services, to identify or sense changes in the context. Then, this information has to be processed by an adaptation reasoning engine to decide which actions should be taken to adapt the mobile application. This adaptation process can help to optimize the use of resources and improve the performance and user experience. For example, by filtering information according to the user’s identity and current location, it is possible to present more relevant information while improving the performance and optimizing the use of resources, given that less memory, processing and network resources are needed as the amount of data sent to the mobile application is reduced. Therefore, it is possible to observe that adaptability is a very important aspect for mobile applications.

2.2.1.2 Resource constraints and limitations

Mobile applications also face challenges related to resource constraints and limitations on the mobile devices. Therefore, it is important to consider aspects like small displays, limited battery life and other limitations like processing power, available memory or storage capacity when designing and
implementing mobile applications (Satyanarayanan, 1996). On the other hand, there are limitations on the communication and data transmission, especially because wireless networks are not as stable as fixed wired networks (Pakkala, et al., 2004) (Qilin, et al., 2009). In this kind of networks, the connection can suddenly be broken or be intermittent. Moreover, the user can change from one network to another due to a change in his location, so it is important to understand that bandwidth, costs and other aspects may vary depending on the network. Finally, a mobile application should be able to handle situations where there is no connectivity at all, while maintaining the application session continuity (Wu, et al., 2010). It is true that many of this constraints and limitations have been reduced in the last years with the advances in mobile technologies; however, the requirements for mobile technologies have increased too.

2.2.1.3 Cross-platform support

Another important challenge of mobile applications is the cross-platform support. Nowadays, there is a wide variety of mobile devices and mobile OSs, and the market is evolving and changing at a rapid pace. Few years ago, Symbian5 and Windows Mobile (now Windows Phone)6 were the leading mobile OSs. However, nowadays iOS7 and Android8 have taken a leading position. Moreover, due to the fast evolution and constant innovation in mobile technologies, it is very difficult, if possible, to predict how the market will look like in few years from now. Likewise, it is very important to consider the diversity of mobile devices during the design and implementation of mobile solutions. Three challenges related to the diversity of mobile devices are recognized in (Bosch, et al., 2007). First, having diverse mobile devices with different hardware and software capabilities. Second, the quick evolution of mobile technologies, and thus the short time a specific mobile device model is in the market. Third, mobile applications have to be tested individually for each type of mobile device. From these challenges is possible to observe the importance of not being limited to one mobile platform. However, in order to provide cross-platform support, it is necessary to consider the additional efforts, skills, and costs required during the design, development, testing, distribution, and operation of mobile applications. Fortunately, nowadays there are different types of tools to support cross-platform development. However, trade-offs between standard solutions and solutions optimized for certain platforms have to be considered.

2.2.2 Enterprise environment considerations

Mobile enterprise applications are used within enterprise environments, thus additional challenges related to enterprises, like heterogeneity, security and IT governance, have to be considered.

2.2.2.1 Heterogeneity

Enterprise system landscapes are characterized by heterogeneity. Enterprises commonly have a diverse software system landscape usually composed by legacy systems, commercial-off-the-shelf software, e.g. Enterprise Resource Planning, Customer Relationship Management or Supply Chain Management, and custom applications (Andersson & Johson, 2001). These complex systems are very important for the business, and it is very difficult and costly to replace or modify them, if possible. Therefore, it is challenging to provide integration between mobile enterprise applications and such heterogeneous landscape of enterprise systems.

5 http://symbian.nokia.com/
6 http://www.microsoft.com/windowsphone/
7 http://www.apple.com/ios/
8 http://www.android.com/
Additionally, integration has to consider other aspects like performance, scalability, security, synchronization and data integrity, and transactional support (Ennai & Bose, 2008) (Pakkala, et al., 2004) (Wu, et al., 2010) (Xue, 2008). Furthermore, current business processes and services need to be adapted and optimized before being used through mobile applications to get the most of them. Optimization is required due to performance issues, resource constraints on the client side, and to provide better context-awareness support.

On the other hand, as connectivity is becoming more reliable, faster and more affordable, it is more feasible to implement online applications. One advantage of this is the reuse of existing business logic from enterprise applications, thus reducing the need of implementing logic on the client side of the mobile application. This also reduces the complexity of the solution, because less synchronization logic is required. However, the offline approach can be preferable for more complex scenarios since data accessibility, user experience, performance and scalability can be improved (Wu, et al., 2010).

Finally, from the user perspective, a uniform and integrated user experience across different types of applications should be provided, i.e. in desktop, Web and mobile applications. Enterprises need to maximize the sharing of services and components while using simple and stable interfaces between common services and platform specific services. The Service Oriented Architecture (SOA) is proposed as a feasible technical solution that facilitates these architectural and integration requirements (Bosch, et al., 2007). This is further discussed in sub-section 3.3.

2.2.2.2 Security

Security is another important aspect to consider in the enterprise environment, and as mentioned before, it is fundamental to enable mobility in the enterprise. Typically, enterprises strive for standardization, control and commoditization of IT services, but usually this is not possible for mobile solutions. Enterprises have to deliver applications and services in an environment they cannot fully control. This brings many risks, especially for the enterprise data. Therefore is important to consider the security mechanisms and policies that are currently in place as they have to be extended or adjusted for the mobile solutions (Pakkala, et al., 2004).

2.2.2.3 IT Governance

Finally, IT governance is an important tool for the management of mobile solutions. IT Governance is defined as:

“*The distribution of IT decision-making rights and responsibilities among enterprise stakeholders, and the procedures and mechanisms for making and monitoring strategic decisions regarding IT.*” (Peterson, 2004)

According to this definition, IT Governance facilitates the management of processes and capabilities associated with IT solutions across the enterprise. Additionally, IT governance is important from the tactical and strategic perspectives, as it can provide accountability and guidance in decision-making processes related to mobile enterprise applications in the medium and long term, e.g. standardization of development processes or technologies (Jones, 2011) (Redman, 2010). The authors of (Bieberstein, et al., 2005) emphasize the importance and benefits of these internal technology standards and methodologies, and discuss how IT governance plays an important role in the decision-making processes of IT initiatives to manage the associated risks.

Nowadays, many enterprises start implementing mobile solutions without establishing first governance policies or having an enterprise mobility strategy. This can result in duplication of efforts, higher costs and implementation times, and reduction of reusability across the portfolio of mobile
Monitoring can be an important tool to identify opportunities for improvement. Likewise, analytics can provide useful information that can later be used in the decision making processes and to establish or adjust policies. Therefore, an additional challenge for mobile enterprise applications is related with the implementation of governance policies as well as monitoring processes.

2.3 Taxonomies of Mobile Enterprise Applications

Nowadays, there are different types of mobile enterprise applications, and the associated challenges vary according to the type of application. Therefore, it is important to understand the characteristics of this kind of applications. Mobile enterprise applications can be classified in several ways. The following classifications are based on the richness and complexity of the mobile application, and on the target group of users.

Five categories of mobile enterprise applications, according to the richness and complexity of the application, are presented in (Unhelkar & Murugesan, 2010). The first category is mobile broadcast applications, where the only intent is to spread information or content to a group of users. The second category is mobile information, where the mobile application also presents read-only information to the user, but in this case the user requests specific information so the information can be personalized for each user. The third category is mobile transaction. As opposed to the previous two categories, the user can perform transactions in this kind of applications, so the communication is no longer one way. The forth category is mobile operation, where the user also performs transactions but with more intensive use of multiple backend systems. The last category is mobile collaboration, where the application supports collaboration and dynamic interactions among users.

On the other hand, mobile enterprise applications can be classified according to the target group of users (Basole, 2007). The mobile enterprise applications are classified as: Business-to-Consumers (B2C), Business-to-Business (B2B), and Business-to-Employees (B2E). B2C mobile applications are focused on engaging and providing services to customers, and their main concerns are related to user experience and scalability. The latter, because the number of users is usually unknown and can grow or change quickly. B2B mobile applications are targeted to partners and suppliers, and their main focus is on providing information or extending services outside the enterprise boundaries. These applications are usually related to supply chains systems or project management systems, and their main concerns are related to security and interoperability. B2E mobile applications are internal applications, i.e. applications targeted to employees of the company. These applications are designed to provide mobile access to enterprise systems and enterprise information, and their main concerns are related to security and integration.

2.4 Mobile Application Architectures

For many people, a mobile application is only the application installed on the end-user mobile device. However, very often these applications need to communicate with enterprise systems to execute some business logic or to exchange data. A mobility-enabled architecture is usually composed by mobile client applications, enterprise systems and a mobile middleware layer that facilitates the communication between the two. In (Tao & Chen, 2010) the authors present a technical architecture for mobile information systems composed by four layers, viz. the presentation layer, the mobile communication layer, the mobile business platform layer and the foundation business layer. The first layer corresponds to the mobile client application. The second is the mobile network which could be considered as part of the public and enterprise network infrastructure. The third layer represents the mobile middleware, and the last one represents the enterprise systems. These same elements are found in (Wu, et al., 2010). Figure 1 shows the elements of a common mobility-enabled architecture:
Figure 1 – Elements of a mobility-enabled architecture (Wu, et al., 2010)

The mobile client application may be deployed in different ways according to the specific requirements of each mobile application. On the other hand, the mobile middleware layer provides different services to support security and facilitate the communication between the mobile client application and the enterprise applications.

2.4.1 Mobile client

There are different approaches for developing and deploying a mobile client application. Nowadays, it is common to focus on three main approaches, viz. native, hybrid, and Web-based. Each approach has advantages and disadvantages, and should be selected according to the specific requirements of the mobile application. The following sub-sections describe each approach separately, followed by a comparison of the three approaches.

2.4.1.1 Native mobile applications

According to the research performed by VDC Research Group⁹ (Krebs & Klein, 2012), native applications are currently the most common approach for developing and deploying mobile client applications. Native applications offer better performance and user experience, and they have direct access to the features and resources on the device. Figure 2 shows the typical architecture of a native mobile application. The limitations for a native application are only related to the capabilities of the

⁹ http://www.vdcresearch.com/
target device and platform. However, the main problem comes when the mobile application has to support multiple platforms, since in that case, the application has to be developed and distributed separately for each platform. Consequently, there is a need for development expertise on each platform, and for a distribution process through different app stores; thus, increasing time and costs.

2.4.1.2 Web-based mobile applications

A Web-based mobile application is another approach for developing and deploying mobile client applications. These applications are executed within the mobile browser and not directly by the mobile OS; thus, the performance is not as good as in native applications. Usually these applications are developed using technologies like HTML5, CSS3 and JavaScript, which are widely supported by current mobile platforms. These technologies are constantly evolving and increasing their support for device features. Figure 3 shows the typical architecture of a Web-based mobile application.

Web-based applications have many advantages. First, and maybe the most important one, is the multiplatform support. With this approach it is possible to have only one application, which can run on multiple platforms. Another benefit is that developers are already familiar with Web technologies, making it easier to develop the mobile applications. Because of these advantages, these applications are less costly to develop and maintain.

On the other hand, this approach presents some disadvantages. The user experience is not as good as with native applications, because Web-based applications have a generic look and feel that is not consistent with native applications installed on the user’s device. Additionally, many features and services exposed by the mobile platform, e.g. camera, push notifications, hardware buttons, etc., are usually not accessible via Web-based applications. Finally, although HTML5 has some support for offline scenarios, usually these applications are only designed for online scenarios.

2.4.1.3 Hybrid mobile applications

The third approach, the hybrid mobile applications, is a combination of the previous two. Hybrid applications have a native component, usually known as “wrapper” or “container”, and on top of it they have an embedded Web-based application. There is a wrapper specific for each supported platform. The wrapper provides APIs to access platform features and services, while

---

10 [http://mobilehtml5.org/](http://mobilehtml5.org/)
the user interface is implemented using standard Web technologies. Therefore, it is possible to have the advantages of native applications and Web-based applications, while reducing the limitations of each approach.

However, the performance and user experience with this approach are not as good as with native applications. Additionally, each platform has its own user interface design guidelines and styles, which are not used because of the generic Web user interface. Nevertheless, the hybrid approach can be extended to provide some user interface elements as part of the wrapper; thus, providing some native elements as part of the user interface and delivering a more consistent layout for each platform. The main design concern is to determine which elements should be delivered natively and which ones through Web technologies. Again, this depends on the specific requirements of the mobile applications. It is important to notice that special skills and expertise are needed to implement native code. Consequently, as more elements are developed natively the reusability of code decreases, and the development and testing costs increase. Figure 4 shows the typical architecture of a hybrid mobile application.

![Figure 4 – Hybrid mobile application architecture (XRG systems Inc, 2011)](image)

**2.4.1.4 Comparison of approaches for mobile clients**

In (Bosch, et al., 2007), the authors present a comparison from the business and technical perspectives between approaches equivalent as the ones discussed above. From the business perspective, they consider the time-to-market and the required investment. From the technical perspective, they consider the user experience, based on ability to access local resources and features, the number of versions needed to support multiple platforms, i.e. portability, and development and distribution efforts.

Common comparisons are based on trade-offs between development process, total cost of ownership, user experience, performance, access to device features, and cross-platform support. Figure 5 shows a comparison between the previous three approaches according to some of these aspects. Once again, it is important to stress that the approach should be selected according to the specific requirements of the mobile application. For that reason, enterprises should be able to support different approaches.
2.4.2 Mobile Middleware

As shown before, mobile enterprise applications are very complex and need to deal with various challenges linked to the mobile and enterprise environments. Different services and technologies can be used to overcome these challenges, but to implement them on each mobile application is costly and increases the complexity of the application. Therefore, these services can be implemented in a middleware layer between the mobile client application and the enterprise systems (Emmerich, 2000). A mobile middleware prevents adding unnecessary complexity at the client side or adding more complexity to current enterprise systems. Additionally, it is responsible for providing a transparency level, as well as standard or well-known interfaces that allow mobile enterprise applications to use the supporting services (Tarkoma, 2009).

In terms of adaptability, the mobile middleware must provide services for context-awareness support. These services have to collect context information from the application or other sources, process this information using an adaptation engine, and then adapt the system as needed (Tarkoma, 2009). For example, a CRM mobile application may use location information to show the user some nearby potential customers. The radius of nearby potential customers could be determined by the density of potential customer near the current location. On the first request, the application retrieves information about all nearby potential customers, while future requests only retrieve information about customers that were not previously retrieved and that are on the radius according to the new location. Additionally, if the application detects there is no connection available, it can switch to offline mode and use previously retrieved information filtered according to the new position. Similarly, the detail level of customer information can be adapted according to the user device’s capabilities, e.g. display size, available memory, or according to the user preferences.

The mobile middleware must also provide services to overcome resource constraints and limitations of mobile devices. The mobile middleware can provide optimization services to reduce the use of resources on the client side. Optimization can be achieved through data transformation, compression, caching, data distribution, etc. For example, the middleware can provide services to transform the data retrieved from enterprise services into a more suitable format for a mobile scenario, e.g. using JSON instead of XML not only reduces the amount of data transferred but also the required processing in the mobile device. The middleware can also offer compression services to reduce the amount of data transmitted. Although this requires more processing on the client side, and therefore more battery
power, it reduces the wireless usage time, which consumes more battery power than the processor. Moreover, process intensive tasks, e.g. complex calculations, can be processed on the server side.

It is very important that the mobile middleware provides appropriate support for heterogeneity and integration. In mobile enterprise scenarios, heterogeneity is on both sides, i.e. on the client side and on the server side. The middleware must provide services to handle this heterogeneity and the means to facilitate the integration between mobile applications and enterprise systems. The middleware needs to provide session and transaction support, as well as support for synchronous and asynchronous communication. In addition, the middleware must provide synchronization services to support offline scenarios.

Finally, the mobile middleware must provide services to establish, implement and monitor security and management policies. The middleware needs to support different authentication and security mechanisms, e.g. Single Sign-On (SSO), and provide support for security at the message and transport levels. For example, encryption can be used at the message level, and SSL can be used to secure communication.

### 2.5 Mobile Application Development Process

Mobile enterprise solutions have a lifecycle consisting of a series of stages, from scope definition to decommissioning. Figure 6 shows Logica’s approach for mobile solutions. This work is mainly focused on the product development and product management stages within this lifecycle, i.e. the mobile application development process.

The mobile application development process is usually divided in four phases, viz. design, build, testing, and distribution. The process is usually iterative and has several iterations before distributing a new version. Each stage can be executed differently according to the type of mobile client application, e.g. a native mobile application is usually distributed through an app store, while a Web-based mobile application may be distributed through a Web site or other mediums. Therefore, it is important to define the type of mobile client application i.e. native, hybrid or Web-based, as early as possible. For native and hybrid applications it is also important to define the target mobile platforms. These aspects will guide many future decisions during the development process. Some mobile OS vendors have published documentation about the application development process, e.g. (Apple, 2012) for iOS.

Additionally, tasks and responsibilities are assigned to different roles during the process to guarantee the quality of the product. One or more of these roles can be executed by an individual or a team.
Some main responsibilities during the mobile application development process are: user experience, coding, security, integration, testing, project management, and product management.

2.5.1 Supporting tools

Two groups of mobile application platforms were usually identified according to the type of applications they intend to support, i.e. Mobile Enterprise Application Platforms (MEAP), and Mobile Consumer Application Platforms (MCAP). The former was targeted to B2E and B2B mobile applications, while the latter was targeted to B2C mobile applications. However, these two groups of mobile application platforms have converged into what Gartner calls a Mobile Application Development Platform (MADP) (Clark, et al., 2012). Therefore, MADPs provide support for B2B, B2C, and B2E mobile applications, which shows the importance of supporting these three types of mobile enterprise applications. According to Gartner, MADP solutions should provide eight capabilities, viz. integrated development environment, application client runtime, enterprise application integration tools, packaged mobile applications or components, multichannel tools or servers, management tools, security support and hosting.

It is important to mention that different types of tools can be used depending on the selected development approach. For native applications, the Software Development Kit (SDK) from the mobile OS vendor is usually used. The SDK is usually composed of an Integrated Development Environment (IDE), an emulator, specific libraries and supporting tools. For native or hybrid applications targeted to multiple mobile platforms, cross-platform tools can offer additional support. Cross-platform tools provide an authoring programming language, e.g. Java, Ruby, C#, JavaScript, so developers code in one programming language. Then, the application can be compiled into a native application for each target mobile operating system i.e. cross-compiling, or can be packed into a native wrapper, i.e. hybrid application. This approach provides several advantages, e.g. it helps reducing the overall development time and effort by focusing on a single code base and reusing current skills of developers. However, it has some disadvantages as well. One disadvantage is that there is always a period of time between the release of new features in a platform SDK and the support of these new features in the cross-platform tool. Moreover, not all the APIs from all the target platforms are supported. Nevertheless, some cross-platform tools offer the possibility to extend the shared code and write native code for specific or advanced functionality. Finally, another disadvantage is the lack of portability of the applications, resulting in a lock-in with the cross-platform tool vendor. A detailed report about cross-compiling tools is presented in (Vision Mobile, 2012), including the advantages and disadvantages of using these tools, and an analysis about some of the main vendors.

2.6 Summary

Enterprise mobility brings a lot of benefits as well as new challenges for businesses. Four main challenges have been identified, viz. privacy, security, management and integration. Privacy is a delicate concern in an enterprise mobility strategy as the boundaries between personal and professional life can be blurred when users use the same device to access personal and enterprise data; thus, a clear separation between personal and enterprise data is needed. Security is one of the main factors that prevent companies from implementing mobile solutions and has to consider confidentiality, integrity, privacy and availability at the client and server sides, as well as on the wireless networks. Management is another important challenge for enterprise mobility; therefore, it is important to define an approach for the management of mobile solutions and identify the skills and capabilities required to support the processes related to enterprise mobility. Finally, integration is required at the various levels of the enterprise architecture and it has to be seen from different perspectives; consequently,
enterprises need to extend their integration architectures and integration strategies to achieve a seamless integration with mobile applications.

Likewise, mobile enterprise applications face challenges related with the mobile and enterprise environments. Three main challenges associated with the mobile environment have been identified, viz. adaptability, resource constraints and the diversity of mobile devices and mobile OSs. Therefore, it is important to consider that these applications may require context-awareness support, cross-platform support or support for offline scenarios, as these aspects may influence trade-off decisions during the design and implementation of mobile solutions. In like manner, three main challenges associated with the enterprise environment have been identified, viz. the heterogeneous landscape of enterprise systems, the extension and adaptation of security mechanisms, and the implementation of governance policies and monitoring processes to support mobile solutions. Furthermore, different types of mobile enterprise applications have been identified and it has been discussed how the associated challenges vary according to the type of application, have an impact on the mobile application development process and on the selection of supporting tools; thus, it is possible to observe the importance of understanding the characteristics of each type of mobile enterprise application.

Finally, mobility-enabled architectures are usually composed by mobile client applications, mobile middleware solutions and enterprise systems. Three main approaches for mobile client applications have been identified, viz. native, hybrid and Web-based. Each approach has advantages and disadvantages, and should be selected according to the specific requirements of the mobile application; therefore, enterprises should be able to support different approaches. It is important to define the type of mobile client application as early as possible, and the target mobile platforms, because this also has an impact on the mobile application development process and the selection of supporting tools. On the other hand, the mobile middleware is responsible of providing various supporting services for the integration between mobile enterprise applications and enterprise systems, which prevents adding unnecessary complexity at the client side or adding more complexity to current enterprise systems. Furthermore, mobile middleware solutions should provide consistent interfaces to expose these services to mobile enterprise applications.

The identified challenges in enterprise mobility, as well as the characteristics and challenges of mobile enterprise applications, give important insights for the design of the reference integration architecture and provide support for many of the assumptions, scenarios, requirements and design decision discussed in Chapter 4. In addition, this chapter helps to identify the various components required in mobile enterprise solutions, their responsibilities and how they support various important aspects related to the identified challenges. Finally, the topics covered in this chapter are the basis for the discussion about mobile enterprise application integration in the next chapter.
Chapter 3 Mobile Enterprise Application Integration

As discussed before, the system landscape of an enterprise is characterised by heterogeneity and the enterprise application portfolio is composed by diverse types of software systems that have not been designed to interact with each other. Additionally, many of these systems are constantly evolving according to the new business requirements. Nevertheless, these systems have to share data as well as functionality to satisfy the business requirements. Therefore, enterprise application integration is required to manage the complexity of interconnecting autonomously designed applications. The following definition of enterprise application integration is presented in (McKeen & Smith, 2002):

“The plans, methods and tools aimed at modernizing, consolidating, and coordinating the computer applications within an enterprise.”

The previous definition is limited to a particular enterprise context. An analysis of enterprise application integration from an inter-organizational and intra-organization perspective is presented in (Erasala & Yen, 2003), where the authors define enterprise application integration as:

“The integration of applications that enables information sharing and business processes, both of which result in efficient operations and flexible delivery of business services to the customer.”

A broader perspective is considered in the taxonomy for enterprise application integration presented in (Irani, et al., 2003); where the authors identify three levels of solutions for enterprise application integration according to the scope and impact of application integration, see Figure 7. The intra-organization application integration is focused on integration of B2E or internal applications. The inter-organization application integration is linked to B2B applications. Finally, the hybrid application integration is concerned on the integration of the previous two to support B2C applications.

The same three levels of enterprise application integration are applicable for mobile enterprise applications. Nevertheless, initiatives to implement mobile enterprise solutions are usually developed in isolation and in a more opportunistic way. This has resulted in point-to-point solutions that are limited in scope and are difficult to reuse and scale, mainly because they have been designed to address the requirements of a specific application.
However, now companies are starting to build more and more mobile applications that have to communicate with their enterprise systems as well as with external applications and services, e.g. cloud services and services from partners or suppliers. Therefore, companies need a more strategic approach to implement mobile solutions more efficiently and in less time. They need to extend their infrastructure to support the new integration requirements associated with mobile enterprise solutions.

3.1 Challenges of enterprise application integration

There are different challenges in enterprise application integration, viz. the various levels of integration, the heterogeneity at each of these levels, the constant changes in the business requirements, and the fast evolution of information and communication technologies. Moreover, besides of these challenges, enterprise application integration solutions have high non-functional requirements, since many business processes rely on them.

According to (Erasala & Yen, 2003), enterprise integration takes place at the physical level, i.e. connecting devices through a computer network, at the application level i.e. integrating different software applications and database systems, and at the process level, i.e. integrating various business processes. Similarly, four target levels of integration are presented in (McKeen & Smith, 2002), viz. data, application, process and inter-organizational. Data integration facilitates sharing data and distributed objects between applications. Application integration facilitates the communication between applications in order to perform a task. Process integration facilitates the coordination of multiple applications to complete a business process. Finally, inter-organizational integration facilitates linking processes beyond the organization boundaries.

On the other hand, enterprise application integration has to deal with heterogeneity at each of these levels (Erasala & Yen, 2003). Interoperability allows autonomous applications to share functionality and interact across these levels in a loosely coupled manner; thus, it is important to achieve application integration. However, there are conceptual, technological and organizational barriers for interoperability (Chen, et al., 2008). Conceptual barrier are related to syntactic and semantic differences that hinder information exchange. Technological barriers are related to architecture and platform differences, e.g. different communication protocols. Finally, organizational barriers are related to incompatibilities in organizational structures. Therefore, it is possible to observe that standards play an important role on overcoming these barriers to provide interoperability.

Finally, enterprise application integration has to provide enough flexibility to deal with constant changes in the business requirements, and thus constant changes in IT requirements to support these new business requirements (Roshen, 2009). Nowadays, businesses need to react accordingly to changes in the market, since business opportunities for new services and product offerings change quickly due to the fast innovation and evolution of technologies. Additionally, business relations with external business can change, and acquisitions or merging businesses are more common these days. Service-oriented architectures can provide the necessary interoperability and flexibility in the integration of applications at different levels (Mahmood, 2007) (Roshen, 2009). Sub-section 3.3.1 discusses this in more detail.

3.2 Challenges of mobile enterprise application integration

In order to understand the challenges associated with the integration between mobile enterprise applications and enterprise systems, it is important to review some of the challenges of distributed systems as they are directly related. Peter Deutsch outlined 7 fallacies of distributed systems in 1994, viz. the network is reliable, latency is zero, bandwidth is infinite, the network is secure, the network
topology does not change, there is one system administrator and transport cost is zero. In 1997, James Gosling added one fallacy, viz. the network is homogeneous (Rotem-Gal-Oz, 2006). These challenges as well as the challenges for mobile enterprise application integration presented in the following subsections have to be considered during the design and implementation of the reference integration architecture for mobile enterprise solutions.

3.2.1 Security support

Security aspects like confidentiality, integrity, privacy and availability are very important for mobile enterprise applications, and each mobile enterprise application can have unique security requirements for each of these aspects. The reference integration architecture should contemplate various possible scenarios to provide the necessary support. Nevertheless, not all of these aspects are relevant to provide seamless integration, as some aspects can be supported at the application or mobile platform level. For example, the protection of enterprise data stored in the mobile device can be provided via a mobile device management solution. The same is true for the separation between enterprise data and personal data. However, some of these aspects are relevant for the reference integration architecture. Because mobile enterprise applications provide access to enterprise data from everywhere, it is important to provide the means to protect the enterprise data outside the enterprise infrastructure boundaries. For this, it is necessary to extend the security model to provide the required security support across the entire portfolio of mobile enterprise applications.

Enterprise data must be accessible only by authorized entities. However, enterprise data has different levels of confidentiality; as a result the security requirements can differ according to the kind of data used by a particular mobile application. For this reason, it is important to guarantee that the involved parties are properly identified and authorized. This requires support of different authentication levels and authentication mechanisms. In some scenarios, it is only necessary to identify the application that is accessing the enterprise data, e.g. a mobile application that consumes enterprise data that is available to everyone, while in other scenarios, it is necessary to authenticate the user to verify his level of access. Furthermore, the authentication can be made via an internal or external identity provider. For example, a B2E mobile application may authenticate user using an internal identity provider, i.e. an identity provider managed by the company. However, a B2B mobile application may use an external identity provider, since it may require to authenticate external users, e.g. employees of a partner organization.

As mentioned previously, one of the fallacies of distributed systems is that the network is not secure. Therefore, it is important to support different mechanisms to protect the data during transmission. According to the confidentiality level of the enterprise data, the data should be protected during transmission at the message level, transport level or both.

Finally, enterprise applications and enterprise data need to be available for users anywhere, anytime. For this reason, it is important to guarantee the availability of mobile enterprise applications by providing protection against malicious attacks and by reducing the dependencies with other applications and services. In order to provide protection against malicious attacks, it is important to reduce the exposed surface of applications and services and provide proper support for validation, exception management, logging and auditing. On the other hand, mobile enterprise applications cannot fully rely on the network connectivity or other applications and services, as these may fail at anytime. Therefore, it is critical that a mobile application does not stop working if the network, an enterprise system or an external service is down. In order to achieve this, mobile applications should not dependent on other applications or services. Although, it is not possible to achieve this for every scenario, some techniques like redundancy, offline support, caching and asynchronous processing can
assist to achieve high availability. Redundancy allows to reduce the downtime, offline support and caching provide access to some information even if a service is down, and asynchronous processing allows to use message brokers to continue receiving messages while a service is down and process them when the service is up and running again.

### 3.2.2 Integration & Interoperability Support

The reference integration architecture should consider aspects directly related to integration and interoperability as it needs to support diverse types of mobile applications and enterprise systems that have not been designed to work together. It has to support B2C, B2E, and B2B mobile applications, developed internally or externally, using different development platforms and different development strategies, i.e. native, Web-based, or hybrid, and targeted to different mobile platforms. Moreover, these mobile applications have to interact with different enterprise systems, e.g. legacy systems, commercial-off-the-shelf software or custom applications, hosted in heterogeneous and distributed environments, e.g. on premise, cloud or managed by a partner organization. As discussed previously, such diversity and heterogeneity at the client and server sides brings several challenges. Therefore, mobile applications and enterprise systems must agree on the interface contracts they are going to use to interact and exchange data. The interface contracts may describe the application protocol used by the parties for the interaction, the expected behaviours associated with the interaction, the available operations, and the semantics for the data and functionality provided. However, mobile enterprise applications and enterprise systems may evolve independently. Therefore, the reference integration architecture should support multiple versions to run in parallel or provide mechanisms to provide tolerance to some changes.

Additionally, the integration strategy must consider how to provide support for offline and online scenarios. This is especially challenging for offline scenarios, as the complexity of the synchronization mechanisms between the mobile application and the enterprise systems may differ according to the type of involved applications and the requirements of the mobile application. Providing offline support is simple in some scenarios, for example when the interaction is one way as data does not need to be synchronized because it is used only for consumption. However, more complex synchronization solutions are required when the interaction is both ways, especially when data can be manipulated simultaneously within the mobile application or through some other application. Some MADP and mobile middleware solutions already provide synchronization services to support complex synchronization scenarios; thus, the reference integration architecture should integrate those services.

One approach focused on data communication and synchronization is Fuego (Tarkoma, et al., 2006). This approach has three main software components, viz. a messaging service, an event service, and a data synchronization service. The former uses a custom serialization format for mobile clients to improve performance and optimize resource utilization on the device and it supports different communication protocols, including HTTP. The message exchange is asynchronous and it implements the event notification pattern, i.e. publisher-subscriber pattern, via its event service. Finally, the data synchronization service uses an optimistic concurrency model and is performed in two stages to minimize latency impact; the first stage synchronizes file metadata and the second stage synchronizes the file content. Figure 8 shows Fuego middleware APIs on the left side and the implemented services on the right side.

This approach is based on asynchronous communication and Web services, which improves scalability and interoperability. Additionally, performance is improved by reducing the number of requests by transferring multiple objects in a single request. One mayor disadvantage is the use of a custom serialization format, because, even though it improves the performance by using a binary format, it
requires custom libraries on the client side to serialize and deserialize messages; thus, these custom libraries need to be implemented for each supported mobile platform. Another problem is the need of data reconciliation as the approach uses an optimistic concurrency model.

A modular and extensible architecture to support online, offline or a mixed communication scheme for mobile applications called Timeless Architecture for Next Generation mObility (TANGO) is presented in (Wu, et al., 2010). TANGO is composed by four main modules distributed between the client and server sides, as shown in the following figure. The main contribution of TANGO is its flexibility, as it supports different configurations according to the requirements of the particular mobile application. It also considers scalability and openness in terms of integration, interoperability and extensibility. For example, the business logic is exposed as a service interface described via a configuration file similar to the Web Service Description Language (WSDL), so the presentation logic does not need to know if the business logic is implemented on the client side or on the server side, or how it is implemented. However, as this approach is mainly focused on providing access to business logic and data for online and offline scenarios, it does not consider any security or performance aspects. Moreover, it does not provide synchronization support, as it leaves this responsibility to the enterprise systems.
A Web service gateway between the mobile application and the enterprise systems is proposed in (Kozel & Slaby, 2007). In this approach, the data transfer between the mobile application and the gateway uses an optimized proprietary protocol, while the gateway and the Web services communicate through the SOAP protocol. However, the approach does not consider aspects like security or cross-platform support, since it is mainly focused on performance. Moreover, it has interoperability limitations due to the use of a proprietary protocol. An alternative is presented in (Tao & Chen, 2010), where the authors present an architecture for service composition to prevent numerous message exchange between the mobile application and the enterprise systems. Again, a major disadvantage of this approach is that it also uses Web services based on SOAP for the communication between the mobile application and the mobile middleware, which has a lot of discovery, packaging and communication requirements (Ennai & Bose, 2008), and requires support for protocols of the Web service stack e.g. WS-Security, on the client side. Additionally, this approach does not consider aspects like context-awareness support or offline support.

It is possible to observe that many of the previous approaches use proprietary protocols for the communication between mobile applications and the mobile middleware to improve performance. However, this reduces interoperability and, since there is a need for cross-platform support nowadays, the libraries to support the proprietary protocol would have to be implemented for each supported mobile platform. Other approaches use heavy protocols like SOAP for the communication between the mobile application and the mobile middleware, which improves interoperability, but hinders performance and uses more resources on the client side. Finally, it is possible to observe the importance of asynchronous communication in mobile applications to improve scalability.

3.2.3 Performance & scalability Support

The reference integration architecture should also consider aspects related to performance and scalability. As mentioned before, performance is a very important factor to deliver a great user experience in mobile applications. Adaptability and context-awareness support can improve performance of mobile enterprise applications, as they can reduce the amount of data exchanged between the mobile application and the enterprise systems. By providing context-awareness support, the mobile applications can consume more relevant data, as it is filtered according to the context of the mobile application. Moreover, optimization techniques like content negotiation and compression can improve performance as well.

CARISMA is a light-weight and context-aware middleware for multimedia applications that addresses issues related to usage of context information for dynamic adaptation of applications and services in mobile environments (Capra, et al., 2003). The authors propose a reflective system that can achieve dynamic adaptation to context changes by exposing and/or changing dynamically its internal behaviour. CARISMA has two main contributions, viz., the use of a bidding auction approach for conflict resolution and the use of reflective techniques together with policies to describe how context changes should be handled. A similar approach is presented in (Costa, et al., 2004), where the authors present a platform based on Web services technology that supports application configuration at runtime and context-awareness support using reflection techniques in the middleware.

Another mobile middleware for context-aware support is CAPNET, although it is targeted to mobile multimedia applications (Davidyuk, et al., 2004). The middleware is implemented as a set of components that offer different services to applications to provide adaptation support. The core components run at the server side, while the rest of the component can run at the client or server side according to the application requirements and availability of resources. The middleware offers support for network transparency via the connectivity management component, asynchronous messaging and
publish/subscribe event management via the messaging component. On the other hand, it offers location transparency via the service discovery component, and storing and management of context information via the other non-core components. Each application defines resource policies, which are used by the middleware for the adaptation. The component-based architecture of the CAPNET middleware, see Figure 10, contributes to provide flexibility and transparency to applications; thus, reducing their complexity and allowing distribution of component between the client and server sides. Additionally, it supports content adaptation according to the availability of resources and device capabilities. However, the CAPNET mobile middleware does not offer support for security or offline scenarios. Moreover, the messaging support is limited to the XML-RPC protocol, and the service discovery to Jini\textsuperscript{11}, which is not widely supported.

\begin{figure}[h]
\centering
\includegraphics[width=0.6\textwidth]{architecture.png}
\caption{The architecture of CAPNET middleware (Davidyuk, et al., 2004)}
\end{figure}

As mentioned in the fallacies of distributed systems, latency is a major performance concern. Caching can alleviate this by reducing the need to exchange data with a remote service. A multi-level caching strategy is important as this reduces the amount of data transferred as well as the load on the server that hosts the remote service. Another strategy is to reduce the number of calls needed to exchange data with enterprise systems. For example, if a mobile application requires data from two different enterprise systems, it is better to request this data to a service that retrieves the necessary data from both enterprise systems and returns a single response to the mobile applications instead of requesting the data directly to each enterprise system. Mobile enterprise applications designed for performance and scalability should support different communication patterns. Therefore, the reference integration architecture should consider the need to support on-demand communication and communication based on subscriptions to events. Moreover, the communication should be stateless to promote scalability, i.e. each request must contain all the necessary information to be processed; thus, allowing enterprise systems to scale out.

MobileSOA is an approach for context-awareness support and composite applications (Ennai & Bose, 2008). It is based on syndication and offers advantages of loose coupling, independence and integration to enterprise systems via stateless service calls. The MobileSOA platform uses a service-

\textsuperscript{11} http://java.net/projects/jini/
oriented architecture based on Web 2.0 technologies and RESTful Web services. As shown in Figure 11, this framework has four major classes of component, viz. Web 2.0 client applications, MobileSOA device layer components, virtualized services, and mobile device platform components. The client applications use the Mailbox interface to interact asynchronously with the MobileSOA platform. The Mailbox interface routes requests to the Service Manager. The Service Manager is responsible for managing the authentication and establishing a secure session between the mobile applications and the enterprise systems. It also provides access to local, remote and ambient services, which are virtualized and made available via URIs. The service virtualization facilitates the service discovery and service binding. Context updates, including contextual policy updates, are managed via the Context Manager component. The Trust Calculator component uses contextual policies to determine the trust level required for using ambient devices or services. The QoS Manager determines if a service can be used according to the quality of service, policies and context information e.g. use a service according to available network connectivity. The Asynchronous Push Proxy provides a mechanism by which push notifications from enterprise systems can be passed to mobile client applications.

The MobileSOA platform supports service composition and facilitates service discovery and service binding by providing a uniform interface for local, remote and ambient services. Additionally, it provides support for context-awareness, policy management, authentication and secure session, and asynchronous communication between the client application and the enterprise systems. However, this approach has some limitations as it is mainly implemented on the client side; thus, it requires more local resources, which can hinder performance, and needs to be implemented for each supported platform. Moreover, this approach does not offer support for offline scenarios as it requires periodic communication with the enterprise systems to maintain the secure session and to update context and service mapping information.
3.3 Enterprise Application Integration Approaches

An architectural integration style is a generalization of previous successful solutions for common problems in the integration of enterprise systems. It is important to mention that architectural integration styles can be combined to better satisfy the integration requirements. Different enterprise integration solutions are presented in (Andersson & Johson, 2001) as architectural integration styles, following the same reusability principle of design patterns. The authors describe several architectural integration styles in terms of the constraints imposed by each style on the base components and their impact on the quality attributes of the resulting system.

In a similar manner, system integration patterns are presented in (Trowbridge, et al., 2004). The authors present patterns to connect layered applications, i.e. integration through the presentation, functional or data layer. The integration pattern has to be selected according to the restrictions imposed by the existing technical architectures of each application. Functional integration is preferred, because it allows components to access data and functionality of other components, in a distributed manner, through well defined interfaces, i.e. through application programming interfaces (APIs). This abstracts the underlying data representation and enables the reuse of implemented business and validation logic, thus providing a more flexible and robust solution. However, sometimes this is not possible due to a lack of APIs, or simply because the functionality and data are not exposed with the required granularity level. Three alternatives for functional integration are presented, i.e. distributed object integration, message-oriented middleware integration, and service-oriented integration.

![Integration patterns and their relationships](image)

Finally, the authors describe different integration topologies that can be used to design an integration architecture. An integration topology describes the channels, collaborations and mechanisms used in
the integration architecture to exchange messages among applications. Three main patterns are presented by the authors, i.e. point-to-point, broker and message bus, as well as the publish/subscribe pattern that can be used with each of the previous three.

Figure 12 shows the different integration patterns and their relationships. Level one shows the different types of integration layers. Level two shows the logical levels to implement integration between applications, as well the alternatives for each one. And, level three shows the integration topologies, as well as some variations, that can be used to define an integration architecture. The design of the integration architecture has to consider these three levels of integration patterns. As mentioned before, functional integration is preferred at the integration layer level. Different approaches for functional integration exist at the logical integration level. These approaches define how to connect applications besides their different technical architectures so they can share data and functionality. Finally, for the integration topology level, different patterns can be combined according to the requirements, existing applications and the context and scope of the integration architecture.

Additionally, it is important to highlight the Gateway pattern presented in (Andersson & Johson, 2001) and (Trowbridge, et al., 2004), and the Façade pattern presented in (Gamma, et al., 1995). These patterns abstract the access to applications into a single interface, which can simplify the development and maintenance processes. These patterns can be useful in the context of mobile enterprise application integration, as they can hide the complexity of accessing enterprise systems and services deployed on premise or in the cloud, by providing a single and uniform interface for all mobile applications. Additionally, they provide independence to mobile application from changes in enterprise systems (Bosch, et al., 2007) (Erl, 2009). Furthermore, security controls, management policies, versioning, error handling, and analytics are easier to implement in a single location, as well as any translation, optimization and adaptation needed between both sides. Thus, the complexity is reduced with a clear separation between the mobile enterprise applications and the enterprise systems. Moreover, this facilitates development, testing and integration efforts for mobile enterprise applications as the mobile teams only have to deal with a standard interface to interact with enterprise systems.

There are different architectural styles that can be used to achieve functional integration in the integration architecture. The following sub-sections present approaches based on the service-oriented and resource-oriented architectural styles.

3.3.1 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural style for developing and integrating enterprise applications. A service-oriented architecture is based on the service-orientation paradigm, which is a design paradigm for the creation of services. The service-orientation paradigm is based on eight design principles, viz. standardized contract, loose coupling, abstraction, reusability, autonomy, statelessness, discoverability, and composability. An extensive catalogue of design patterns for service-oriented architectures is presented in (Erl, 2009). However, before further discussing service-oriented architectures, it is necessary to have a clear understanding of what a service is. The World Wide Web Consortium12 (W3C) defines a service as:

“An abstract resource that represents a capability of performing tasks that represents a coherent functionality from the point of view of provider entities and requester entities.
To be used, a service must be realized by a concrete provider agent.” (W3C, 2004)

12 http://www.w3.org/
Two important concepts of services are presented in this definition, i.e. service provider, service consumer. Similarly, the Organization for the Advancement of Structured Information Standards\(^\text{13}\) (OASIS) defines a service as:

“A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.” (OASIS, 2006)

This definition highlights the concept of service description to define the interface to access the service. The Open Group\(^\text{14}\) describe a service as a self-contained logical representation of a repeatable business activity that encapsulates its functionality, has a specified outcome, and may be composed of other services (The Open Group, 2012). This definition introduces the concept of service composition. It is possible to identify two key characteristics of services from the previous definitions. First, a service provides a well defined functionality. Second, this functionality can be accessed through a defined interface. Therefore, the service consumer does not need to know the internal implementation details and data structures of the service.

SOA and the service-based integration are the result of the evolution of previous technologies like Sockets for data sharing and connectivity, Remote Procedure Calls for sharing functionality and interface description, Distributed Objects for code reusability and Message-Oriented Middleware for synchronous and asynchronous messaging (Roshen, 2009). Service-oriented architectures are typically implemented through services with a message API style, and using Web service technologies like SOAP, WDSL, and UDDI (Umapathy & Purao, 2010); however, different service API styles may be used (Daigneau, 2011). Web service technologies support three key operations for SOA, i.e. service description, service discovery, and service invocation. The service description is usually implemented through WSDL. It contains information about the functionality provided by the service and the service interface and it is used for the service binding. The service discovery is supported by publishing the service description into a service directory according to the UDDI standard. Finally, SOAP is used for the service invocation. Nevertheless, there are other Web service protocols that provide support for composition, transaction, security, messaging reliability, etc. (W3C, 2004).

A variation of SOA is the event-driven SOA. This approach is a combination of SOA and event-oriented architecture (EDA). It uses a publish-subscribe mechanism to send messages or notifications about events from a publisher system to a subscribed system. SOA and EDA are orthogonal and complement each other, and combined they provide benefits like greater flexibility, loose coupling and improved responsiveness (Edwards, 2007) (Levina & Stantchev, 2009). In the context of mobile solutions, the asynchronous nature of event-driven SOA is very useful as mobile applications can receive notifications about changes on the state of a business process; thus, accelerating the reaction and the decision-making processes.

To sum up, the goal for SOA and Web services technologies is to enable integration between heterogeneous middleware technology stacks, while preserving the low coupling between service consumer and service provider (Mahmood, 2007) (Pautasso, et al., 2008).

### 3.3.2 Resource-Oriented Architecture

A Resource-Oriented Architecture (ROA) is based on resources and the principles of the Representational State Transfer (REST) architecture style (Fielding, 2000). A resource may have one

\(^{13}\) [http://www.oasis-open.org/](http://www.oasis-open.org/)

\(^{14}\) [http://www.opengroup.org/](http://www.opengroup.org/)
or more identifiers, and may have one or more representations of its state at certain point in time. However, an identifier may only identify one resource. In the context of ROA, representations are exchanged between client and services to transfer the state of a resource.

On the other hand, REST is based on six design constraints, viz. client-server, statelessness, cache, layered system, code-on-demand, and uniform interface. These constraints define the interaction between a client and a server for transferring representation of resources. The first constraint follows the separation of concerns principle providing independence between client and server. The second constraint defines a statelessness communication between client and server to improve scalability. The third constraint determines that every response needs to explicitly indicate if it is cacheable or not. The layered system constraint defines that each component can only interact with an immediate layer. The code-on-demand is an optional constraint that allows to extent the client functionality through scripting code, e.g. JavaScript. Finally, the uniform interface defines a common interaction between components, and it is based on four interface constraints, viz. identification of resources, manipulation of resources through representations, self-descriptive messages, and hypermedia as the engine for application state.

A ROA has four basic properties based on the principles presented above, viz. addressability, statelessness, connectedness and uniform interface (Richardson & Ruby, 2007). Addressability is concerned to the exposition and reference to resources and their state through their unique identifiers. Statelessness refers that every request happens in isolation, i.e. requests are idempotent. The client is responsible for maintaining the application state, thus every request needs to contain all the necessary information to be processed by the server. The server is only responsible for maintaining the state of the resource. A resource as well as its state can be represented in different ways. The connectedness refers to the quality of having links, within the resource representation, to navigate to other resource states or other resources. Finally, the uniform interface establishes the use of a defined set of operations to manipulate resources.

A ROA can be implemented through RESTful Web services as discussed in (Richardson & Ruby, 2007). RESTful Web services are services implemented using HTTP as an application protocol and following the principles of the REST architectural style. There has been a lot of controversy about these constraints for implementing RESTful Web Services, especially about the uniform interface constraint and in particular the last constraint for the uniform interface, i.e. hypermedia as the engine for application state. As a result, Leonard Richardson developed the maturity model that describes four levels of maturity for implementing HTTP services using the REST principles (Richardson, 2008). Level 0 uses HTTP only as a transport protocol for remote interactions through the exchange of messages with a singular endpoint, e.g. regular Web services. Level 1 uses URIs to identify resources and allow direct interactions with them. This is equivalent to the first interface constraint in REST, i.e. identification of resources. In this level, instead of sending request to a singular endpoint, the requests are sent to individual resources. Level 2 uses HTTP as an application protocol to better define the semantics for the interactions with each resource. This is equivalent to the REST interface constraint of self-descriptive messages and allows to take advantage of some features of the protocol, e.g. status codes, caching support, etc. Finally, level 3 uses hypermedia controls to define relations between resources so that the representation of a resource contains relations or links to other resources. This is equivalent to the last REST interface constraint, i.e. hypermedia as the engine of application state. Consequently, a RESTful Web service is in level 3. Finally, it is important to stress that, although the author explains the implementation of these levels through Web technologies like HTTP, URIs and hypermedia, REST is not tied to any technology, as it is an architectural style.
In the context of mobile solutions, these RESTful Web services seem to be the preferred solution to access enterprise systems; however, they are not typically used between enterprise systems, as the requirements are different. In (Pérez, et al., 2011), the authors present a model-driven approach, and propose the introduction of an Application Façade Component Model to extend current Web methods to support RESTful Web services and resource-oriented architectures. The main goal of their approach is to facilitate and support the creation of RESTful Web services. However, their approach is limited as it does not cover other aspects related to the consumption and management of these services e.g. security concerns or integration with services outside the boundaries of an organization.

3.3.3 Web-Oriented Architecture

In (Thies & Vossen, 2008) the authors present the Web-Oriented Architecture (WOA) as a subset of service-oriented architectures based on Web 2.0 technologies, e.g. HTTP, SSL, and JSON. They argue that Web-oriented architectures can be more suitable in simpler scenarios, where the complexity of a common implementation of a SOA is not necessary, e.g. based on SOAP, WSDL, and UDDI.

In (Thies & Vossen, 2009), the same authors expand this concept by placing WOA between SOA and the ROA, and proposing three levels for Web-oriented architectures. This approach is based on Web APIs, or Web Procedures as called by the authors. A Web API is a service implemented through Web technologies. In this approach, the core level of a WOA uses a Web API which only uses the HTTP and its GET verb with additional parameters to perform requests. Some people classify this type of services as Low REST Web services (Pérez, et al., 2011). However, as shown in Figure 13, the WOA can be extended to use different technical implementations of a service, i.e. regular Web services, RESTful Web services or Low REST Web services.

![Figure 13 – Classification of WOAs (Thies & Vossen, 2009)](image)

As shown in Figure 14, a Web-oriented architecture has a topology composed by nodes that represent Web Procedures (Web APIs), and Web Architecture Controllers (WAC). A WAC works as a broker to connect Web Procedures, and provides core services through an API, i.e. connectivity management, transformation, process engine, security, governance and monitoring, and integration with Web Procedures, other Web Architecture Controllers and other kind of systems.
This approach provides a link between the previous two approaches and it provides flexibility in terms of the technologies used for the implementation of Web APIs. Such flexibility is important for the integration of mobile applications and enterprise systems, as more appropriate technologies can be used according to the integration requirements. For example, RESTful Web services can be used to support integration with mobile applications, while regular Web services can be used to integrate enterprise systems. Moreover, the WAC can be designed according to the Façade pattern; thus, providing the benefits presented previously.

### 3.4 Web technologies and Web-based APIs

After reviewing different architecture approaches for functional integration, it is time to see how these architectures can be implemented. Enterprises are already following service-oriented architectures approaches and using Web technologies, i.e. SOAP, WSDL and other Web service protocols, to integrate their systems due to the interoperability they can provide. Now, they want to reuse these services with mobile client applications. However, these services have been designed for integration between applications that run on the server side or are part of a wired network of a company, and not for scenarios where resources are scarce and connectivity is not always stable like in mobile scenarios. Therefore, the challenge is to extend the service-oriented architectures currently implemented to provide integration with mobile applications. Although these services can be reused in mobile solutions, they should be first optimized to adapt to the constraints of mobile clients. For this purpose, enterprises can use other Web technologies to develop optimized APIs for mobile applications that can run on top of the current Web services; thus, providing the required optimization, while reusing the existing services.

As discussed above, different approaches and technologies can be used to achieve functional integration. A comparison between RESTful Web services and regular Web services is presented in (Pautasso, et al., 2008). The authors compare the two approaches in terms of architectural principles, conceptual design aspects, and technology options, and they considers the number of architectural decisions and the number of alternatives available for each aspect. From the comparison, the authors conclude that RESTful Web services are more suitable for simpler integration scenarios with less QoS requirements, while regular Web services provide a more robust solution. Another comparison between Web 2.0 technologies, and service-oriented architectures based on SOAP and WSDL is presented in (Schroth & Janner, 2007). The authors compare both approaches from the business and technological perspectives, and present the similarities and differences between the two approaches. The authors conclude that Web 2.0 technologies provide a lightweight solution compared to a service-
oriented architecture that relies on more complex standards, i.e. SOAP and WSDL. However, they consider both approaches complementary and suggest a unification that will facilitate a multichannel access and consumption of Web-based resources.

In the context of mobile applications, the integration between mobile enterprise applications and enterprise systems can be implemented via resource-oriented APIs based on Web technologies like HTTP, URIs, media types, etc. HTTP can be used as an application protocol, providing a uniform interface to manage resources through the semantics of the HTTP verbs and HTTP status codes. Moreover, HTTP can provide support for compression, caching, basic authentication, content negotiation, etc. via the HTTP headers. Finally, HTTP can provide security at the transport level in combination with SSL, i.e. HTTPS. The URIs can be used to identify the resources, while the media types can be used to define the representations used to manipulate the state of resources. These technologies provide the means to satisfy many of the integration requirements in the context of mobile enterprise applications, and without adding any significant infrastructure requirement on the client or server sides.

The design of these Web-based APIs is very important, as they are the link between the client and the server sides. Different challenges are identified in the design of Web-based APIs, e.g. the API structure, composition of services, and optimization for mobile clients. Additionally, when choosing the implementation technologies for the API, it is important to consider possible limitations on the client side e.g. level of HTTP support in certain mobile OS. A data-driven approach for the design of Web-based services for mobile applications is presented in (Riva & Laitkorpi, 2007). The design process is divided into three steps, viz. developing the abstract data model, deriving the core resource model, and designing the URI space of the API. The first step aims to identify the data entities that will be part of the API, the type of entity, i.e. individual item or collection of items, and their associations. The second step classifies these entities into primary and secondary resources. And the third step defines the structure of the API, i.e. URI space, the allowed methods and their semantics, the available content types, the representation of a resource, and the link structure between resources. However, this approach does not specify how to handle more complex requirements in the URI space of the API, e.g. versioning, pagination, data field customization, message enrichment, etc.

Besides of the aspects considered so far, there are other aspects that need to be considered in order to provide seamless integration between mobile enterprise application and enterprise systems, e.g. transformation, transaction, synchronization, caching, authorization, composition, etc. Web-based APIs only facilitate this integration by providing standard and easy to consume interfaces to mobile client applications. Therefore, an integration architecture and other supporting technologies are needed to satisfy all the integration requirements of mobile enterprise applications.

### 3.5 Summary

Initiatives to implement mobile enterprise solutions are usually developed in isolation and in a more opportunistic way, which has resulted in point-to-point solutions that are limited in scope and are difficult to reuse and scale as they have been designed to address requirements of specific applications. Now, companies need a more strategic approach as they are starting to implement more mobile solutions; thus, they need to extend their infrastructure to support the new integration requirements associated with mobile enterprise solutions. Consequently, enterprise application integration is required to manage the complexity of interconnecting these applications since they have not been designed to interact with each other. However, there are different challenges related to enterprise application integration, viz. integration at the data, application, process and inter-organizational levels, heterogeneity at each of these levels, constant changes in the business requirements, and fast evolution
of information and communication technologies. Therefore, it is necessary to consider integration at the data, application, process and inter-organizational levels, and use standards for the implementation of service-oriented architectures to provide the necessary interoperability and flexibility in the integration of applications at these levels.

On the other hand, the challenges for mobile enterprise application integration are directly related to challenges of distributed systems; therefore, it is important to understand the required support for security, integration, interoperability, performance and scalability in mobile enterprise solutions. In addition, it is important to recognize that not all of these aspects are relevant for the reference integration architecture; thus, some of them can be implemented at the application level. Nevertheless, many of these aspects are already supported by specialized services; thus, the reference integration architecture should consider the integration with these services.

Additionally, different approaches for enterprise application integration have been discussed and it has been observed that functional integration is preferred, because it allows components to access data and functionality of other components, in a distributed manner, through well defined interfaces. In this regard, the Gateway and the Façade patterns are useful in the context of mobile enterprise application integration, as they can hide the complexity of accessing enterprise systems and services deployed on premise or in the cloud, by providing a consistent interface for all mobile applications, while providing evolution independence between mobile applications and enterprise systems. Furthermore, security and management controls are easier to implement in a single location, as well as any translation, optimization and adaptation needed between both sides. Moreover, this facilitates development, testing and integration efforts for mobile enterprise applications as the mobile teams only have to deal with a standard interface to interact with enterprise systems.

Finally, different integration architectural styles have been discussed, including SOA, ROA and WOA, to understand how their principles and constraints can be applied in the context of mobile enterprise application integration. Thus, the challenge is to extend current implementations of service-oriented architectures to provide integration with mobile applications. For this purpose, enterprises can use other Web technologies to develop optimized APIs for mobile applications that can run on top of the current Web services; thus, providing the required optimization, while reusing the existing services.
Part II: Reference Integration Architecture
Chapter 4  Reference Integration Architecture Design

As stated before, the goal of this master thesis is to define a reference integration architecture for mobile enterprise solutions. Therefore, it is important to have a better understanding of what is a reference architecture and what is an integration architecture. A definition of reference architecture is:

“A reference architecture is, in essence, a predefined architectural pattern, or set of patterns, possibly, partially or completely instantiated, designed and proven for use in particular business and technical contexts, together with supporting artifacts to enable their use. Often, these artifacts are harvested from previous projects.” (Kruchten, 2000)

From this definition it is possible to observe that a reference architecture is targeted to a particular context, and it is composed by a number of components based on architectural and design patterns. On the hand, a definition of integration architecture is:

“The technology architecture of two or more connected applications or systems including whatever technologies, resources, or extensions were added to enable their integration. Many integration architectures include middleware platforms and associated adapter or bridging extensions.” (Erl, 2009)

Therefore, a reference integration architecture is composed by a number of components that enable integration between two or more connected applications, arranged according to some architectural and design patterns, and targeted to a particular context, e.g. mobile enterprise solutions. Therefore, the reference integration architecture should describe the components, as well as the architectural and design patterns used to define its distribution, required to facilitate the integration between mobile enterprise applications and enterprise systems.

The reference integration architecture might be used as a tool to identify integration requirements between mobile enterprise applications and enterprise systems, and to guide decisions during the design and development of mobile enterprise solutions. The reference integration architecture is not intended to be a complete solution; on the contrary, it is only a starting point to facilitate the design and implementation of mobile solutions within a company. In this way, enterprise architects and software architects do not need to start a design from zero and can reuse solutions from previous applications with similar requirements. The reference integration architecture should provide the means to facilitate information exchange between mobile enterprise applications and enterprise systems, and be flexible enough to support requirements from different types of mobile applications.

Additionally, the reference integration architecture can be used during the selection of a mobile application development platform (a.k.a. MADP) or other mobile middleware solution. It may also guide decisions during design and development of particular mobile enterprise applications. The reference integration architecture considers the main and most common integration requirements for mobile solutions; however, each company and each mobile solution may have additional requirements. Therefore, the reference integration architecture could be extended to satisfy the specific requirements of a company or of a particular mobile solution.

To summarize, the reference integration architecture can be used as a baseline for the initial design, as a tool to identify integration requirements, and to identify elements required for a particular mobile solution. It is based on some assumptions and common scenarios of mobile enterprise applications. A list of requirements and constraints have been extracted from these scenarios and assumptions, as well as from the challenges identified in the research phase and described in the previous chapters.
4.1 Assumptions

The design of the reference integration architecture assumes that the company has an Enterprise Mobility Strategy (EMS) already defined, where the different types of mobile applications that need to be supported, the scope of mobile solutions within the company, and some constraints for those solutions are specified. The design also assumes there are three security levels of information:

- **Public**: Information that is available to everyone.
- **Private**: Non-public information that requires authentication and authorization mechanisms to protect the access to it.
- **Confidential**: Sensitive information that requires authentication, authorization and higher security mechanisms to protect the access to it.

Finally, the design assumes that the organization has a service-oriented architecture approach to provide functional integration and interoperability across the enterprise application portfolio. Additional assumptions that have been made during the design of the reference integration architecture are briefly described in the following table.

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Description</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-platform support</td>
<td>The EMS requires to support mobile enterprise applications for multiple mobile OSs.</td>
<td>▪ The variety of mobile devices and mobile OSs in a constantly evolving market.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Companies are implementing BYOD initiatives.</td>
</tr>
<tr>
<td>Diversity of mobile clients</td>
<td>The EMS requires to support native, hybrid and Web-based mobile client applications.</td>
<td>▪ All approaches for developing and deploying mobile client applications have advantages and disadvantages.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ An approach must be selected according to the specific requirements of a mobile application.</td>
</tr>
<tr>
<td>Diversity of mobile middleware solutions</td>
<td>The EMS requires to support different mobile application development platforms and mobile middleware solutions.</td>
<td>▪ Mobile applications may be developed internally or externally according to their requirements and the available skills and capabilities in the company.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Complex mobile solutions may require specific mobile middleware solutions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Mobile middleware solutions may be deployed on premise or in the cloud.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Description</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different types of mobile enterprise applications</td>
<td>The EMS requires to support B2B, B2E and B2C mobile applications.</td>
<td>▪ Enterprises aim to increase support for mobile solutions and provide more B2E mobile applications.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Enterprises may use B2C or B2B mobile applications to provide information to customers or partners.</td>
</tr>
<tr>
<td>No point-to-point mobile solutions</td>
<td>According to the EMS, no point-to-point mobile solutions should be implemented, i.e. mobile applications should not communicate directly with enterprise systems.</td>
<td>▪ It is more difficult and expensive to maintain, manage and secure point-to-point solutions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Mobile applications and enterprise systems may evolve independently; thus, there must be a layer of indirection between the two.</td>
</tr>
<tr>
<td>Diversity of mobility management solutions</td>
<td>According to the EMS, some aspects related to enterprise mobility should be controlled by specialized mobility management solutions.</td>
<td>▪ Enterprise mobility has an impact on different levels and processes of a company, and it is constantly evolving; thus, no solution covers all the aspects of enterprise mobility management.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Enterprise mobility requirements are specific to one company; thus, a commercial-off-the-shelf solution may not satisfy all of these requirements.</td>
</tr>
</tbody>
</table>

*Table 1 – Assumption for the design of the reference integration architecture.*
4.2 Scenarios

Scenarios describe situations that a system is likely to face during operation, along with the expected behaviour or response from it (Rozanski & Woods, 2005). Therefore, scenarios help identifying the requirements and constraints of a particular system. For that reason, different scenarios have been used to identify some of the requirements and constraints for the reference integration architecture. These scenarios are a result of the research phase, where various common scenarios for mobile enterprise applications have been identified with the collaboration of people from the mobile development teams within Logica.

These following scenarios aim to describe the common interactions between mobile enterprise applications, enterprise applications, and other required infrastructure services through a Façade layer. Each scenario contains a brief description, a list of considerations, a list of conditions, a flow of events describing the expected system response, a list of related scenarios and a sequence diagram.

4.2.1 Verify mobile enterprise application identity

<table>
<thead>
<tr>
<th>Description</th>
<th>The Façade layer has to verify the identity of a mobile enterprise application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Considerations</td>
<td>▪ The authentication mechanism is based on the claims-based identity model.</td>
</tr>
</tbody>
</table>
| Conditions | ▪ The request contains information to identify the mobile enterprise application.  
▪ The mobile enterprise application is registered with the identity provider. |
| Flow of events | 1. The Façade layer receives the request and extracts the information to identify the mobile enterprise application from the request.  
2. The Façade layer sends the information to the identity provider to identify the mobile enterprise application.  
3a) If the verification process succeeds, the Façade layer continues processing the request.  
b) If the verification process fails, the Façade layer interrupts the processing of the request and sends an error message to the mobile enterprise application. |

<table>
<thead>
<tr>
<th>Rel. scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile Enterprise Application</td>
</tr>
<tr>
<td>1. Request + security token</td>
</tr>
<tr>
<td>Response</td>
</tr>
</tbody>
</table>

4.2.2 Authentication with internal identity provider

<table>
<thead>
<tr>
<th>Description</th>
<th>The Façade layer has to authenticate the user of a mobile enterprise application using an internal identity provider.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Considerations</td>
<td>▪ The authentication mechanism is based on the claims-based identity model.</td>
</tr>
</tbody>
</table>
| Conditions | ▪ The request does not contain a security token.  
▪ The user is registered in the internal identity provider.  
▪ The Façade layer has a trust relation with the internal identity provider. |
| Flow of events | 1. The Façade layer receives the request and determines that an authentication
The process is needed.
2. The Façade layer sends a response to the mobile enterprise application to redirect it to the appropriate identification provider.
3. The user enters his credentials and the mobile enterprise application sends the user’s credentials to the identity provider.
4. The identity provider authenticates the user using the provided credentials, generates a security token and sends it to the mobile enterprise application.
5. The mobile enterprise application receives the security token and stores it securely for future requests.
6. The mobile enterprise application sends a request to the Façade layer, along with the security token received from the identity provider.
7. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.

<table>
<thead>
<tr>
<th>Rel. scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile Enterprise Application</td>
</tr>
<tr>
<td>1. Request</td>
</tr>
<tr>
<td>Response</td>
</tr>
</tbody>
</table>

### 4.2.3 Authentication with external identity provider to access internal service

<table>
<thead>
<tr>
<th>Description</th>
<th>The Façade layer has to authenticate the user of a mobile enterprise application using an external identity provider.</th>
</tr>
</thead>
</table>
| Considerations | - This kind of authentication may be required in B2B mobile applications to authenticate a user from a supplier or partner organization.  
- This kind of authentication may be required in B2C mobile applications to authenticate a user using his accounts from an internet service provider, e.g. Facebook, Google or Windows Live.  
- The authentication mechanism is based on the claims-based identity model.  
- The authentication with external identity providers is based on authentication delegation. |
| Conditions | - The request does not contain a security token.  
- The user is registered with the external identity provider.  
- The Façade layer has a trust relation with the external identity provider. |
| Flow of events | 1. The Façade layer receives the request and determines that an authentication process is needed.  
2. The Façade layer sends a response to the mobile enterprise application to redirect it to the appropriate identification provider.  
3. The user enters his credentials and the mobile enterprise application sends the user’s credentials to the identity provider. |
4. The identity provider authenticates the user using the provided credentials, generates a security token and sends it to the mobile enterprise application.
5. The mobile enterprise application receives the security token and stores it securely for future requests.
6. The mobile enterprise application sends a request to the Façade layer, along with the security token received from the identity provider.
7. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.

Rel. scenarios

<table>
<thead>
<tr>
<th>Mobile Enterprise Application</th>
<th>Façade layer</th>
<th>External : Identity Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Request</td>
<td>2. Unauthorized</td>
<td>3. Credentials</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Security token</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Stores security token</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6. Request + security token</td>
</tr>
<tr>
<td>Response</td>
<td></td>
<td>7. Process request</td>
</tr>
</tbody>
</table>

4.2.4 Authentication with external identity provider to access external service

**Description**
The Façade layer has to authenticate the user of a mobile enterprise application using an external identity provider.

**Considerations**
- This kind of authentication may be required in mobile enterprise applications that consume external services that have their own identity providers.
- This kind of authentication may be required in mobile enterprise applications to authenticate a user using his accounts from an internet service provider, e.g. Facebook, Google or Windows Live.
- The authentication mechanism is based on the claims-based identity model.
- The authentication with external identity providers is based on authentication delegation.

**Conditions**
- The request does not contain a security token.
- The user is registered with the external identity provider.
- The Façade layer has a trust relation with the external identity provider.

**Flow of events**
1. The Façade layer receives the request and determines that an authentication process is needed.
2. The Façade layer sends a response to the mobile enterprise application to redirect it to the appropriate identification provider.
3. The user enters his credentials and the mobile enterprise application sends the user credentials to the identity provider.
4. The identity provider authenticates the user using the provided credentials, generates an authorization code and sends it to the mobile enterprise application.
5. The mobile enterprise application receives the authentication code and sends a request to the Façade layer, along with the authentication code.
6. The Façade layer receives the request, extracts the authentication code from the...
request and sends it to the identity provider.
7. The identity provider verifies the authentication code, generates a security token and sends the security token to the Façade layer.
8. The Façade layer receives the security token and stores it securely for future requests.
9. The Façade layer continues processing the request by sending a request to the external service, along with the security token received from the external identity provider.

<table>
<thead>
<tr>
<th>Mobile Enterprise Application</th>
<th>Façade layer</th>
<th>External : Identity Provider</th>
<th>External Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Request</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Unauthorized</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Credentials</td>
<td></td>
<td>Validate Credentials</td>
<td></td>
</tr>
<tr>
<td>4. Authentication code</td>
<td></td>
<td>Validate authentication code</td>
<td></td>
</tr>
<tr>
<td>5. Request + authentication code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Authentication code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Security token</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Stores security token</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Request + security token</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**4.2.5 Request public information**

**Description**
A mobile enterprise application sends a request to the Façade layer to get public information from an enterprise system.

**Considerations**
- As the requested information is public, it is only necessary to identify the mobile enterprise application.
- The response may only contain a partial set of the requested data if data pagination is supported for the type of request.

**Conditions**
- The request is well formed.
- The request contains information to identify the mobile enterprise application.
- The mobile enterprise application has access to the requested information.
- The enterprise system is available and the requested information exists.

**Flow of events**
1. The Façade layer receives the request and verifies the identity of the mobile enterprise application as in scenario 4.2.1
2. The Façade layer establishes a connection with the enterprise system and retrieves the requested information.
3. The Façade layer sends the response synchronously according to the request parameters.

**Rel. scenarios**
Scenarios 4.2.1, 4.2.8, 4.2.9
4.2.6 Request private information

Description: A mobile enterprise application sends a request to the Façade layer to get private information from an enterprise system.

Considerations:
- As the requested information is private, it is necessary to authenticate the user and verify if he has access to the requested information.
- The response may only contain a partial set of the requested data if data pagination is supported for the type of request.

Conditions:
- The request is well formed.
- The request contains the required security token and the security token is valid.
- The enterprise system is available and the requested information exists.

Required system response:
1. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.
2. The Façade layer establishes a connection with the enterprise system and retrieves the requested information.
3. The Façade layer sends the response synchronously according to the request parameters.

Rel. scenarios: Scenarios 4.2.2, 4.2.3, 4.2.4, 4.2.8, 4.2.9

4.2.7 Request confidential information

Description: A mobile enterprise application sends a request to the Façade layer to get confidential information from an enterprise system.

Considerations:
- As the requested information is confidential, it is necessary to authenticate the user, verify if he has access to the requested information and use additional security mechanisms to protect the information during transmission.
- The Façade layer is responsible for providing the security mechanisms to protect the information during transmission. The mobile enterprise application or other mobile management solutions should provide the mechanisms to secure the information in the mobile application and mobile device.
- The response may only contain a partial set of the requested data if data pagination is supported for the type of request.
pagination is supported for the type of request.

**Conditions**
- The request is well formed.
- The request contains the required security token and the security token is valid.
- The mobile enterprise application sends a request using a secure channel.
- The mobile enterprise application has a valid key to decrypt the information in the response.
- The enterprise system is available and the requested information exists.

**Required system response**
1. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.
2. The Façade layer establishes a connection with the enterprise system and retrieves the requested information.
3. The Façade layer encrypts the information and performs auditing tasks.
4. The Façade layer sends the response synchronously according to the request parameters and through the secure channel.

**Rel. scenarios**
Scenario 4.2.2

### 4.2.8 Writing in the cache of the Façade layer

**Description**
The Façade layer writes data from a response on its cache for future similar requests.

**Considerations**
- The request is well formed.
- The information is not confidential.
- The information from the response has been successfully retrieved from the enterprise application.

**Conditions**
- The request is well formed.
- The information is not confidential.
- The information from the response has been successfully retrieved from the enterprise application.

**Required system response**
1. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.
2. The Façade layer establishes a connection with the enterprise system and retrieves the requested information.
3. The Façade layer writes the retrieved information on its cache to serve future similar requests.
4. The Façade layer sends the response synchronously according to the request parameters.

**Rel. scenarios**
### 4.2.9 Reading from the cache of the Façade layer

**Description**  
The Façade layer reads data from its cache to serve a request.

<table>
<thead>
<tr>
<th>Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>- The request is well formed.</td>
</tr>
<tr>
<td>- The information is not confidential.</td>
</tr>
<tr>
<td>- The requested information is on the cache of the Façade layer.</td>
</tr>
</tbody>
</table>

**Required system response**

1. The Façade layer receives a request to get information from an enterprise system.
2. The Façade layer verifies if it can serve the request with information from its cache.
3. The Façade layer reads the requested information from its cache and sends the response synchronously according to the request parameters.

**Rel. scenarios**

### 4.2.10 Send push notification

**Description**  
The Façade layer sends a push notification to a registered mobile device when occurs a type of event that the user of the mobile device is subscribed to.

**Considerations**

- The Façade layer can delegate the responsibility of sending push notifications to another service from a mobile middleware solution if exists.

**Conditions**

- The user is subscribed to the type of event.
- Push notifications are enabled for the mobile enterprise application in the mobile device.

**Required system response**

1. The Façade layer receives a notification of an event from an enterprise system.
2. The Façade layer verifies which users are subscribed to the event.
3. The Façade layer sends a push notification to the mobile device of the user, along with information about the event.

**Rel. scenarios**
4.2.11 Request information considering current context

<table>
<thead>
<tr>
<th>Description</th>
<th>A mobile enterprise application sends a request to the Façade layer to get information from an enterprise system according to its current context.</th>
</tr>
</thead>
</table>
| Considerations | • The context-awareness support is based on the application-aware strategy, i.e. the mobile application and the Façade layer collaborate to provide context-awareness support (Satyanarayanan, 1996).  
  • The context information from the request may be used by the Façade layer to adapt the process of the request, e.g. using compression depending on the network connectivity of the mobile client, adapting the response according to the capabilities of the mobile device, etc. |
| Conditions | • The request is well formed.  
  • The request contains the required access control information and it is valid.  
  • The request contains context information captured by the mobile enterprise application.  
  • The Façade layer may delegate the context-awareness support to another service from a mobile middleware solution if exists  
  • The enterprise system is available and the requested information exists. |
| Required system response | 1. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.  
  2. The Façade layer extracts the context information from the request and uses that information to process the request.  
  3. The Façade layer establishes a connection with the enterprise system and retrieves the requested information.  
  4. The Façade layer adapts the retrieved information according to the context information obtained from the request.  
  5. The Façade layer sends the response synchronously according to the request parameters. |
| Rel. scenarios | Scenarios 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.8, 4.2.9 |
4.2.12 Request information from multiple sources

**Description**
A mobile enterprise application sends a request to the Façade layer to get information from multiple enterprise applications.

**Considerations**
- The response might only contain a partial set of the requested data if data pagination is supported for the type of request.

**Conditions**
- The request is well formed.
- The request contains the required access control information and it is valid.
- The enterprise systems are available and the requested information exists.

**Required system response**
1. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.
2. The Façade layer establishes a connection with each involved enterprise system and retrieves the requested information.
3. The Façade layer collects the retrieved information and performs the required transformations and mapping to the data transfer object for the response.
4. The Façade layer sends the response synchronously according to the request parameters.

**Rel. scenarios**
Scenarios 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.8, 4.2.9

4.2.13 Request synchronous processing

**Description**
A mobile enterprise application sends a request to the Façade layer to create, modify or delete information in an enterprise system. This request must be processed synchronously.
Considerations
- The request might not contain data on its body according to the type of operation requested.
- The response might not contain data on its body according to the type of operation requested.

Conditions
- The request is well formed.
- The request contains the required access control information and it is valid.
- The enterprise system is available.

Required system response
1. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.
2. The Façade layer validates the received data if any.
3. The Façade layer establishes a connection with the enterprise system and process the requested operation.
4. a) If the processing succeeds, the Façade layer sends a response synchronously.
   b) If the processing failed, the Façade layer performs the required exception management and sends a response synchronously along with a description of the error.

Rel. scenarios
Scenarios 4.2.1, 4.2.2, 4.2.3, 4.2.4

4.2.14 Request asynchronous processing

Description
A mobile enterprise application sends a request to the Façade layer to create, modify or delete information in an enterprise system. This request must be processed asynchronously.

Considerations
- The request might not contain data on its body according to the type of operation requested.
- The Façade layer can delegate the responsibility of processing the requests in the queue to another service if exists.
- A push notification may be sent to the mobile device of the user after the
information has been processed in the enterprise application.

**Conditions**
- The request is well formed.
- The request contains the required access control information and it is valid.
- The enterprise system may not be available.
- The information processing on the enterprise system may be executed by the Façade layer or other component.

**Required system response**
1. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.
2. The Façade layer validates the received data if any.
3. The Façade layer inserts the request into a queue for further processing.
4. The Façade layer sends a response synchronously, acknowledging to have received the request.

**Rel. scenarios** Scenarios 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.10

**4.2.15 Refresh information from enterprise system**

**Description**
A mobile enterprise application sends a request to the Façade layer to verify it is has the latest information from an enterprise system.

**Considerations**

**Conditions**
- The request is well formed.
- The request contains the required access control information and it is valid.
- The request contains information about the last time the mobile enterprise application received data from the Façade layer.

**Required system response**
1. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.
2. The Façade layer verifies in its cache if the information has changed.
3. a) If the information has changed, the Façade layer sends the response synchronously with the information from its cache.
   b) If the information has not changed but it has expired:
      b1. The Façade layer establishes a connection with the enterprise application and retrieves the requested information.
      b2. The Façade layer writes the information on its cache.
      b3. The Façade layer sends the response synchronously.

**Rel. scenarios** Scenarios 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.8, 4.2.9
### 4.2.16 Synchronize information with enterprise system

**Description**
A mobile enterprise application sends a request to the Façade layer to synchronize information with an enterprise application.

**Considerations**
- The Façade layer delegates the responsibility for synchronization to another service.

**Conditions**
- The request is well formed.
- The request contains the required access control information and it is valid.

**Required system response**
1. The Façade layer receives the request, extracts the security token from the request, verifies its signature and continues processing the request.
2. The Façade layer forwards the request to a specialized service.
3. The Façade layer sends a response synchronously, acknowledging to have received the request.

**Rel. scenarios**
Scenarios 4.2.1, 4.2.2, 4.2.3, 4.2.4
4.3 Requirements

The requirements for the reference integration architecture have been identified based on the challenges for enterprise mobility, mobile enterprise applications and mobile enterprise application integration identified in the research phase, and from the scenarios presented previously.

4.3.1 Security requirements

As security is one of the most important barriers for enterprise mobility, the reference integration architecture should provide proper support to facilitate the implementation of security mechanisms. In addition, it should aim to reduce the exposed surface of enterprise systems in order to reduce related security risks and vulnerabilities. Therefore, a layer to manage the access and interactions between mobile enterprise applications and enterprise systems should be added, and direct access to enterprise systems should not be allowed.

Privacy support, i.e. a clear separation between enterprise data and personal data in the mobile device and within the mobile enterprise applications, is usually managed via specialized solutions, e.g. virtualization. Consequently, the reference integration architecture does not need to consider privacy aspects within its design. Similarly, the security on the client side, i.e. on the mobile device and mobile enterprise application, is usually managed through MDM or MAM solutions. Therefore, the reference integration architecture does not need to consider security requirements on the client side. However, it should be possible to exchange information with the mobile management solutions in order to be consistent with the established security and governance policies.

On the other hand, the reference integration architecture should consider security requirements on the server side. It should support different authentication processes according to the particular requirements of the mobile solution. Among others, it should support:

- Use of an access key to identify a particular mobile enterprise application.
- Authentication with internal identity provider to access internal services.
- Authentication with external identity provider to access internal services.
- Authentication with external identity provider to access external services

Additionally, the reference integration architecture should support authentication and authorization with various identity providers. Furthermore, it must provide support for validation, exception management, logging, and auditing. This is very important, since all data must be validated before being processed by any enterprise system. Additionally, if any error occurs during the processing of a request, it must be handled properly, e.g. by logging detailed information of the error, handling the exception, and sending only the essential error information to the mobile enterprise application.

Finally, the reference integration architecture should provide support to secure enterprise information during transmission across wireless networks, according to the required security level for the enterprise information. For this purpose, it should provide support to secure the data at the message and transport levels. At the message level, it must provide encryption support, i.e. support to encrypt a message, to decrypt a message, and to configure and access encryption keys or certificates. At the transport level, it must provide support to secure the transport channel using technologies like SSL.

A list of Security Requirements (SSR) for the reference integration architecture is presented below:

SSR1. Integration with mobile management solutions to exchange information about security and governance policies.
SSR2. Support different authentication processes.
SSR3. Support authentication and authorization with various identity providers.
SSR4. Provide support for securing enterprise data at the message and transport levels.
SSR5. Provide support for validation, exception management, logging, and auditing.

4.3.2 Integration and interoperability requirements

Since organizations are implementing BYOD initiatives and the market of mobile devices is constantly evolving, it is important for the reference integration architecture to be agnostic to any mobile platform. Similarly, as enterprises realize the value and benefits of enterprise mobility for their business, they are going to increase their support for mobile solutions. Therefore, the reference integration architecture should support B2C, B2E, and B2B mobile applications, whereas they have been developed internally or externally, using different mobile development platforms, or as native, hybrid or Web-based mobile applications. Furthermore, the reference integration architecture should provide integration with different enterprise systems managed by the organization or by a partner organization, and hosted on premise or in the cloud. Consequently, it should use standard technologies to provide interoperability in such heterogeneous environment.

Additionally, the reference integration architecture should aim to reduce the dependencies between mobile enterprise applications and enterprise systems. By doing this, the availability of mobile enterprise applications can increase, as they will continue working even if some enterprise systems are not. Moreover, this allows mobile enterprise applications and enterprise systems to evolve independently. Finally, this also promotes scalability as requests for processing information in enterprise systems can be processed asynchronously.

Finally, the reference integration architecture should facilitate context-awareness support, advanced synchronization for offline scenarios, and event-oriented integration via push notifications. For this purpose, the reference integration architecture should be able to integrate with mobile middleware solutions that provide these kinds of specialized services.

A list of integration and interoperability requirements (IIR) for the reference integration architecture is presented below:

IIR1. Agnostic to any mobile platform.
IIR2. Support B2C, B2E, and B2B mobile applications, developed internally or externally, as native, Web-based, or hybrid mobile applications, and using different mobile development platforms.
IIR3. Provide integration with different enterprise systems managed by the organization or by a partner organization, and hosted on premise or in the cloud.
IIR4. Reduce dependencies between mobile enterprise applications and enterprise systems to increase availability, independent evolution, and promote scalability.
IIR5. Provide integration with specialized services from mobile middleware solutions to offer context-awareness support, synchronization support for offline scenarios, and event-oriented integration via push notifications.

4.3.3 Performance and scalability requirements

Performance and scalability are very important for mobile enterprise applications, due to the resource constraints in mobile environments and the increasing number of mobile applications and mobile users. Caching is a very common solution to improve performance and scalability for applications that principally consume information like mobile enterprise applications do. There are different strategies for caching, e.g. using an expiration strategy to eliminate the need to send requests in many cases or
using a validation strategy to eliminate the need to send full responses in other cases. Additionally, caching can be supported at various levels, e.g. at the client or at an intermediary. Thus, the reference integration architecture must support different caching strategies and various levels of caching.

The reference integration architecture should also provide the means for mobile enterprise applications and enterprise systems to negotiate how to exchange data, e.g. via content negotiation or supporting synchronous and asynchronous processing of requests. This way, the mobile enterprise application can request information in a particular format and according to its capabilities and available resources.

An important aspect for performance and scalability is the ability to monitor a system. Thus, the reference integration architecture must provide the means to monitor the performance and usage of the different services as well as the number of interactions between mobile enterprise applications and enterprise systems. This information might be useful to identify and predict peak loads and traffic in order to plan the required resources to support them. Additionally, this information may be used to identify opportunities for performance improvements.

A list of performance and scalability requirements (PSR) for the reference integration architecture is presented below:

PSR1. Support different caching strategies and various levels of caching.
PSR2. Provide mechanisms to support content negotiation and processing type negotiation between mobile enterprise applications and enterprise systems.
PSR3. Provide the means to monitor the performance and usage of the different services as well as the number of interactions between mobile enterprise applications and enterprise systems.

4.4 Components and relationships

As mentioned before, the reference integration architecture should describe the components required to facilitate the integration between mobile enterprise applications and enterprise systems, as well as the architectural and design patterns used to define their distribution. Hence, the reference integration architecture is composed by mobile enterprise applications, mobile application development platform or mobile middleware solutions, a façade layer, enterprise systems and infrastructure services. The relations of these components are shown in Figure 15 and described in the following sub-sections.

![Figure 15 – Components of the reference integration architecture](image-url)
4.4.1 Mobile enterprise applications

The reference integration architecture has to support different types of mobile enterprise applications, including B2B, B2C, and B2E. Additionally, it has to support diverse mobile clients, including native, hybrid and Web-based. Finally, it should support mobile enterprise applications targeted to multiple mobile OSs and developed using diverse Mobile Application Development Platforms (MADP).

Mobile enterprise applications exchange messages with the Façade layer to access data and functionality from enterprise systems. Similarly, they may use services from mobile application development platforms or mobile middleware solutions, as well as some infrastructure services, to accomplish some task or support some functionality.

4.4.2 Mobile Application Development Platforms or Mobile Middleware

The reference integration architecture has to support different MADP or mobile middleware solutions, because complex scenarios may require the use of services from these solutions. Similarly, this is true when the organization does not have the required skills and capabilities required for the design and implementation of a mobile enterprise solution; thus, it delegates this to another organization and a MADP or mobile middleware may be required for the solution.

4.4.3 Façade Layer

The Façade layer is one of the most important components of the reference integration architecture, as it facilitates the integration between mobile enterprise applications and enterprise systems by providing a consistent, simple and extensible interface between them. Furthermore, the Façade layer may use services from MADP, mobile middleware solutions, mobile management solutions or other infrastructure services.

4.4.4 Enterprise Systems

The reference integration architecture has to provide integration with different types of enterprise systems and services, including legacy systems, databases, commercial-off-the-shelf applications, custom applications, B2B services and cloud services.

4.4.5 Infrastructure Services

The reference integration architecture has to use various infrastructure services to provide support for security, monitoring, management. For example, it may require using identity providers for authentication. Similarly, it may require to access information from mobile management solutions to verify if the device or application that is requesting enterprise data has access to it or if it complains with the established policies.

4.5 Design decisions

This sub-section discusses the main design decision for the reference integration architecture. It starts by listing the design principles used for the design of the reference integration architecture.

4.5.1 Design principles

The following principles have been used to guide decisions during the design of the reference integration architecture:
• Provide a consistent interface to access enterprise systems from mobile enterprise applications independently of the type of mobile application, the type of mobile client, the target mobile platform or the platform used to develop the mobile application.

• Provide a simple and easy-to-learn interface that allows internal or external developers to use it without knowing the internal implementation details.

• Provide an architecture open for extension but closed for modifications that may break clients, i.e. mobile enterprise applications.

• Provide a clear separation between mobile enterprise applications and enterprise systems, so that they can evolve independently.

• Provide an architecture based on open standards to increase interoperability and guarantee a larger audience.

4.5.2 Architectural styles

A goal of the reference integration architecture is to provide integration between mobile enterprise applications and enterprise systems at the functional level, i.e. allowing mobile enterprise applications to reuse implemented business and validation logic from enterprise systems through well defined interfaces. Three alternatives for functional integration are presented in sub-section 3.3, including service-oriented integration. Additionally, as mentioned before, the design of the reference integration architecture assumes that the organization has a service oriented architecture approach to provide functional integration and interoperability across the enterprise application portfolio. Consequently, the reference integration architecture is based on the service-oriented architecture style to provide integration with mobile enterprise applications and enterprise systems. However, it must provide a consistent, simple and easy-to-use interface for the mobile enterprise applications; therefore, the integration with mobile enterprise applications is also based on the resource-oriented architecture style. Finally, Web technologies should be used in the reference integration architecture, as it should be based on open standards to increase interoperability and guarantee a larger audience. Thus, the integration with mobile enterprise applications is also based on the Web-oriented architecture style.

As discussed in sub-section 3.3, each architectural style imposes constraints to achieve certain quality attributes for the resulting system. In this case, the service-oriented architecture style facilitates integration by implementing standardized contracts, promoting reusability and reducing the coupling between components. Similarly, the resource-oriented architecture style promotes simplicity and independent evolution by providing a uniform interface for communication and data exchange, and it improves scalability by adding cache support and not storing state information in the server side. Performance is also improved with the cache support, and it is ideal for mobile applications as it helps to reduce latency and the amount of transmitted data. Finally, the Web-oriented architecture style promotes the use of Web 2.0 technologies like HTTP, SSL, and JSON, which are more suitable for simpler scenarios e.g. mobile applications. Nowadays, these technologies are pervasive, i.e. are supported by almost any platform, allowing to support a wide range of clients.

4.5.3 Integration topology

Once the main architectural styles have been chosen, it is time to define the integration topology for the reference integration architecture. As mentioned in sub-section 3.3, the integration topology defines the channels, collaborations, and mechanisms to exchange messages used by the components. The decisions related with these aspects are described in the following sub-sections.
4.5.3.1 Message broker
As discussed in sub-section 3.3.3, a Web-oriented architecture topology based on a Message Broker, (see Figure 14), can provide several core services through an API. Similarly, the Façade layer should provide additional services while it relays messages between mobile enterprise applications and enterprise systems. This layer is based on the Façade pattern discussed in sub-section 3.3. This pattern abstracts the access to applications into a single and consistent interface; thus, hiding the complexity of accessing heterogeneous enterprise systems independently of where they are deployed. Moreover, it provides a clear separation between the mobile enterprise applications and the service providers, i.e. enterprise systems, infrastructure services, mobile middleware solutions, etc., and allows them to evolve independently. Finally, this pattern reduces development, testing and integration efforts for mobile enterprise applications, as mobile development teams only have to employ a standard interface to interact with enterprise systems.

4.5.3.2 Service API styles
Another important decision for the reference integration architecture is related to the API style for the services. There are three different API styles for services, i.e. Remote Procedure Call API or RPC APIs, Message API, and Resource API. RPC APIs are simple to implement; however, the client applications are directly coupled to the remote procedures exposed by a service. Consequently, an independent evolution between mobile enterprise application and enterprise systems is not possible. For that reason, this style of service APIs is not used in the reference integration architecture.

Message APIs use self-descriptive messages that contain information about the remote procedure that need to be used to serve the request. Thus, a layer of indirection between client and the remote procedure is added. However, there is still a coupling between the message and the remote procedure, as the message structure needs to be changed if an argument needs to be added, changed or removed from the remote procedure. Additionally, the data is wrapped within message envelopes which increases the size of the messages. Finally, this approach can lead to a proliferation of message descriptions, as two messages have to be defined for each operation, i.e. request message and response message. Nevertheless, this API style supports late binding, synchronous or asynchronous processing, different message encodings, and can support caching; however, specialized infrastructure is needed for that. Since message APIs are commonly used in service-oriented architectures implementations, the Façade layer should access enterprise systems through message APIs when they are available.

Finally, resource APIs use service contracts composed by an application protocol, resource identifiers and media types. Usually, they are implemented using HTTP as the application protocol, URIs to identify resources, and standard media types, thus being a good choice for scenarios that need to support a wide variety of clients. The advantage is that clients do not need to learn specialized APIs, as the interactions use predefined semantics of the application protocol. This API style also supports asynchronous and synchronous processing, different message encodings, and can provide support for late binding by adding links to related resources within the representation in the response. An additional advantage of using media types is that there is no need to use message envelopes to wrap the data as the media type defines how to process the resource. This reduces the amount of data transferred and the processing needed, resulting in better performance. Moreover, content negotiation allows clients to define their preferences about representation of a resource for each request. Finally, this approach is ideal for information consumption scenarios like in mobile applications. And when used with HTTP, it can take advantage of caching technologies specially designed for this protocol, without adding any infrastructure requirements. Due to the above, the Façade layer of the reference
integration architecture must expose resource APIs to mobile enterprise applications based on HTTP and other Web technologies.

4.5.4 Façade layer

As discussed previously, the Façade layer is a very important component in the reference integration architecture. This sub-section describes many decisions related to its design.

4.5.4.1 Deployment environment

An important decision is where to deploy the Façade layer, on premise or in the cloud. Mobile enterprise applications have to be accessed from everywhere and it could be difficult to predict the number of users, e.g. in B2C mobile applications. The cloud provides a flexible infrastructure, i.e. the infrastructure enables enterprises to easily scale up or down the hosted applications. Additionally, cloud providers offer caching services that can be used to reduce latency by having data geographically distributed as shown in the following figure.
Similarly, as mobile applications may access other cloud services, it may reduce latency times if the services are deployed within the same data centre of the cloud provider. Finally, the infrastructure of cloud providers offers the necessary support for high availability as they have Service Level Agreements with their customers. For these reasons, a cloud deployment is more suitable for the Façade layer. Of course there may be additional benefits related with initial infrastructure investments and operational costs; however, they have not been considered here.

4.5.4.2 Service Interceptor

According to the requirements presented above, the reference integration architecture needs to support different authentication processes with various identity providers, as well as provide support for exception management, logging, monitoring, and auditing. As these aspects are used across various services, the Service Interceptor pattern presented in (Daigneau, 2011) is used. This pattern permits to encapsulate cross-cutting concerns like authentication, logging, etc., so that they can be loaded into a pipeline that will be called after receiving a request and before sending a response. Therefore, the Service Interceptor pattern can be combined with the Pipes and Filters pattern (Buschmann, et al., 1996). In this case, an incoming pipeline is defined with various Service Interceptors that are called in a predefined sequence after receiving a request. Similarly, an outgoing pipeline is defined with various Service Interceptors that are called in a predefined sequence before sending a response. This pattern promotes maintainability by allowing to reuse the service interceptor across various services; thus, reducing the duplication of code. Furthermore, service interceptor may be used to support transformations before the request is processed by a service and after is the response is sent, e.g. providing support for encoding/decoding, encryption/decryption. The following figure shows the combination of the service interceptor pattern with ingoing and outgoing pipelines between the client and the service.

![Service Interceptor pattern with ingoing and outgoing pipelines](Image)

*Figure 18 – Service Interceptor pattern with ingoing and outgoing pipelines*
4.5.4.3 Front Controller

After a request goes through the ingoing pipeline, it has to be sent to the right service to be further processed. In this case, the Front Controller pattern (Fowler, 2003), is used as a centralized point for routing requests to the right service. The request is routed to a service based on the URI and HTTP verb of the request, and then the Front Controller uses a routing table and these values to select the service. The routing table contains information about registered service controllers, supported HTTP verbs and URI templates. Moreover, the Front Controller can perform additional validations around the URI of the request, e.g. validate which HTTP verbs are supported by the URI, or validate the URI with respect to the URI template and its associated constraints.

4.5.4.4 Services

Now it is time to specify the general design of the services that are responsible of processing the requests. These services are called by the Front Controller and have two main components, viz. the service interface and the service controller.

As the name suggests, the service interface defines the interface or contract of the service, i.e. the formats of the incoming and outgoing messages of the service. The service interface is composed by Data Transfer Objects, Request and Response mappers, and Media Type formatters. Data Transfer Objects, a.k.a. DTOs, define the structure of the data contained in request or response messages associated with a service. These DTOs are used with Media Type formatters to support content negotiation, as different media type formatters can define how to handle different representation’s formats of a DTO, i.e. a media type formatter can provide support for reading and/or writing different DTOs. Finally, the request mappers may be used to map request DTOs to domain entity objects of service providers, while response mappers may be used to map domain entity objects of service providers to response DTOs. In the case of response mappers, they can incorporate hypermedia controls to the response DTOs through the Linked Service pattern (Daigneau, 2011). Consequently, it is also possible to provide support for the hypermedia as the engine of application state (HATEOAS) constraint of REST.

It is important to stress that DTOs should not follow an inside-in design, i.e. based on the data exposed by the service providers. Instead, DTOs should follow an outside-in design approach, i.e. based on the data required by the mobile enterprise applications. This is very important because a proper design can reduce the number of requests from a mobile enterprise application to obtain the required information. Finally, DTOs should use collections instead of single records or items. This allows to process multiple items within one request, reducing the number of required requests and thus improving the overall performance (Buschmann, et al., 2007).

As the name suggests, service controllers are based on the Service Controller pattern (Daigneau, 2011). They contain methods to process requests according to the result of the routing from the Front Controller. A service controller has a service interface defined as described above. Each method of a service controller contains all the necessary logic to process a particular type of request, e.g. validation of received data, orchestration between the services from service providers that are required to produce a response, creation of responses, etc. The service controller uses service connectors to handle the communication logic to consume services from the service providers. The following figure shows an overview of the design of these services.
4.5.4.5 Mobile clients

According to the identified requirements, the reference integration architecture has to support diverse mobile clients, i.e. mobile enterprise applications, and be agnostic to any mobile platform. Therefore, the communication between the mobile clients and the Façade layer is based on Web technologies. In particular, HTTP is used as an application protocol to consume the Web APIs exposed by the Façade layer. Therefore, mobile clients only need a HTTP client to consume services exposed by the Façade layer. However, mobile clients may require to manage some common tasks, e.g. set HTTP headers, apply transformations to the message payload, attach security tokens, handle exceptions, etc., in order to use services from the Façade layer. For that reason, the Service Gateway pattern, which is based on the Gateway pattern (Fowler, 2003), should be used to encapsulate these tasks from the mobile application.
The service gateway must have the required information to communicate with the Façade layer, e.g. it can obtain this information from a configuration file. Before sending a request, the service gateway may perform tasks like validation, serialization, encoding, encryption, attach security tokens, or even verify if the request can be served from a local cache. After receiving a response, the service gateway may perform tasks like validation, de-serialization, decoding, or decryption. Finally, it is important to highlight the fact that the service gateway has to be implemented for each supported mobile platform; however, the benefits of reusability are relevant if there are at least two mobile applications that consume services from the Web API for a mobile platform.

4.5.4.6 Service providers

As explained before, the design of the reference integration architecture assumes a service-oriented architecture approach in the enterprise to provide functional integration and interoperability across the enterprise application portfolio; therefore, the diverse enterprise systems can be accessed through services. The same is true for external services, e.g. services from partner organizations, and cloud services used by the mobile enterprise applications. Therefore, as mentioned before, service connectors are used to handle the communication logic to consume services from the service providers. Service connectors encapsulate the access to services from different service providers, and permits to expose a standard interface to the service controllers.

According to the type of services exposed by a service provider, a service connector may be implemented using the Service Proxy pattern (Gamma, et al., 1995), the Service Gateway pattern (Fowler, 2003), or the Channel Factory (Gamma, et al., 1995). Service Proxies are used with services that expose a message API, e.g. SOAP Web services, while Service Gateways are used with services that expose a resource API, e.g. HTTP services. Finally, Channel Factories are used for infrastructure services.

![Diagram](image.png)

Figure 21 – Relation between service connectors and service providers.

To provide an overview, the following figure shows the overall design for the Façade layer and its related components.
Figure 22 – Overall design of the Façade layer and its related components.
4.5.5 Claim-Based Identity

As discussed in 4.3.1, the reference integration architecture should be able to support different authentication processes as well as authentication and authorization with various identity providers. The claim-based identity model allows to support federated identity while leaving authentication logic outside applications. Therefore, applications do not need to implement logic to determine the identity of a user or application; instead, applications receive claims about a user or an application issued by a trusted authority. The main concepts of the claim-based identity model are described in Table 2, while Figure 23 shows the implementation of federated identity using the claim-based identity model.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>An application component that invokes Web services or issues HTTP requests on behalf of a local user.</td>
</tr>
<tr>
<td>Federation Provider</td>
<td>A type of identity provider that provides single sign-on functionality between an organization and other identity providers and relying parties.</td>
</tr>
<tr>
<td>Identity Provider</td>
<td>An organization issuing claims in security tokens, a.k.a. issuer.</td>
</tr>
<tr>
<td>Relaying Party</td>
<td>A claim-based application that relies on security tokens and claims issued by an identity provider.</td>
</tr>
<tr>
<td>Security Token</td>
<td>An on-the-wire representation of claims that has been cryptographically signed by the issuer of the claims, providing strong proof to any relying party as to the integrity of the claims and the identity of the issuer.</td>
</tr>
<tr>
<td>Security Token Service</td>
<td>A claims provider implemented as a Web service that issues security tokens.</td>
</tr>
</tbody>
</table>

Table 2 – Main concepts of the claim-based identity model (Baier, et al., 2011).

Figure 23 – Federated identity using a claim-based identity model. Adapted from (Chappell, 2011)
4.6 Extension points

As discussed previously, a reference integration architecture is composed by a number of elements that enable integration between two or more connected applications, arranged according to some architectural and design patterns, and targeted to a particular context. In this chapter, a reference integration architecture for mobile enterprise solutions has been presented. This reference integration architecture is based on some assumptions and common scenarios of mobile enterprise applications. The scenarios and the requirements described in this chapter are the result of the research phase and are associated with the challenges of enterprise mobility, mobile enterprise applications and mobile enterprise application integration discussed in chapters 2 and 3. Moreover, this chapter describes the components of the reference integration architecture, their distribution and their relationships, as well as the main decisions during its design.

The proposed reference integration architecture can be used as a baseline for the design and implementation of mobile solution, as a tool to identify integration requirements between mobile enterprise applications and enterprise systems, and to identify elements required for a particular mobile solution, e.g. mobile application development platforms, mobile middleware solutions or mobile management solutions. As mentioned before, the reference integration architecture is not intended to be a complete solution; therefore, the reference integration architecture has the following extension points:

- **Service interceptors**: It is possible to add or remove service interceptors in the ingoing or outgoing pipelines.
- **Services**: It is possible to add new services within the Façade layer to support new services from service providers. For example, to support a new service from a partner organization.
- **Media types**: It is possible to add new media types for a particular DTO or for all the services within the Façade layer. For example, to support a proprietary media type.
- **Service providers**: It is possible to add new service providers to the Façade layer. For example, a new enterprise system.
- **Identity providers**: It is possible to add support for new identity providers by adding an identity provider into the list of trusted identity providers of the federation provider.
- **Cloud provider services**: It is possible to add support for additional infrastructure services from the cloud provider. For example adding caching support at the Façade layer.
Chapter 5  Integration Architecture Implementation

For the implementation phase, an exploration of available technologies, frameworks and tools has been carried out in order to implement the reference integration architecture. Then, the designed architecture has been implemented using the selected technologies, frameworks and tools.

This chapter discusses the implementation of an integration architecture based on the design presented in Chapter 4. It starts by describing the scope and limitations of the implementation. Then it explains the frameworks and tools used for the implementation, and it presents some examples. Finally, it discusses the results of the implementation as well as some observations.

5.1 Implementation scope and limitations

As discussed in sub-section 4.4, the proposed reference integration architecture is based on various components. However, it has not been possible to consider all these components within the implementation of the integration architecture, mainly because of limited resources in terms of time and money. Therefore, some components have not been considered due to software licenses required to use them, due to the infrastructure required to install them, or due to the time required to learn how to use them. Because of these circumstances, the implementation has the following limitations on the components of the proposed reference integration architecture:

- **Mobile enterprise applications:** The implementation does not include mobile enterprise applications developed with cross-platform tools. It only includes mobile enterprise applications developed with the SDKs from mobile OS vendors. Additionally, the mobile enterprise applications are not deployed to mobile devices; instead, emulators have been used. The implementation details for mobile enterprise applications are further explained in the next chapter.

- **Mobile application development platforms (MADP) or mobile middleware solutions:** The implementation does not include any MADP or mobile middleware solution.

- **The façade layer:** It is one of the most important elements of the reference integration architecture; therefore, the implementation does not have limitations related with it. The Façade layer is deployed in the cloud using a 90-day free trial subscription from a cloud provider.

- **Enterprise systems:** The implementation does not include real enterprise systems; instead, it uses enterprise data services to emulate them. Additionally, the implementation only includes public cloud services that do not have any fees associated with their usage. The implementation details for enterprise data services and the use of cloud services is further explained in the next chapter.

- **Infrastructure services:** The implementation only includes infrastructures services available through the cloud provider used. However, there are additional limitations for the use of those infrastructure services since a trial subscription has been used for the implementation.

Because of these limitations, the implementation of the reference integration architecture does not consider the components crossed out in Figure 24:
5.2 Frameworks and tools

The implementation of the reference integration architecture is mainly based on Microsoft technologies because the master thesis was carried out within the Microsoft practice at Logica. Additionally, as Logica is partner with Microsoft, it was easier to get some tools required for the implementation, e.g. Visual Studio or SQL Server Management Studio. Some of the frameworks and tools used for the implementation are listed below:

- .NET Framework 4.0
- ASP.NET Web API
- Enterprise Library 5.0 – May 2011
- Fiddler Web Debugger
- Thinktecture IdentityModel 4.0
- SQL Database
- SQL Server 2012
- SQL Server Management Studio
- Visual Studio 2010
- Windows Azure Access Control
- Windows Azure Caching
- Windows Azure Cloud Services
- Windows Azure Queues
- Windows Azure SDK for .NET – June 2012
- Windows Azure Service Bus
- Windows Identity Foundation
- Windows Identity Foundation SDK
- Windows Communication Foundation
5.2.1 ASP.NET and ASP.NET Web API

On May 31, 2012, Microsoft\(^\text{15}\) announced the Release Candidate version of ASP.NET MVC 4.0, which included the ASP.NET Web API among the new features delivered in this version. ASP.NET Web API is a framework for creating HTTP services using the .NET Framework (Microsoft, 2012). This framework contains a HTTP programming model that facilitates the manipulation of data related with HTTP requests and HTTP responses. It is worth mentioning that Henrik Frystyk Nielsen, one of the principal authors of the HTTP 1.1 specification (IETF, 1999), has participated in the design of this framework. The framework also provides support for content negotiation, and partial support for the Open Data Protocol, a.k.a. OData\(^\text{16}\).

The ASP.NET Web API framework has been used for the implementation of the Façade layer, because it has been built based on technologies used within the ASP.NET framework. Therefore, it inherits or has access to many features from the ASP.NET framework. For example, ASP.NET provides support to implement the Front Controller pattern (Microsoft, 2003), routing based on URIs (Microsoft, 2010), and the Service Interceptor pattern through Intercepting Filters (Microsoft, 2003). Nevertheless, it is important to describe some important classes within the ASP.NET Web API before explaining more details about the implementation. The following table list these classes and presents a description based on the documentation from (Microsoft, 2012):

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApiController</td>
<td>Defines properties and methods for API controller. It implements the IHttpController interface. Some important members are: Configuration, ControllerContext, Request, Url.</td>
</tr>
<tr>
<td>DelegatingHandler</td>
<td>A base type for HTTP handlers that delegate the processing of HTTP response messages to another handler, called the inner handler. It inherits from the HttpResponseMessage class.</td>
</tr>
<tr>
<td>HttpClient</td>
<td>Provides a base class for sending HTTP requests and receiving HTTP responses from a resource identified by a URI.</td>
</tr>
<tr>
<td>HttpConfiguration</td>
<td>Configuration of HttpServer instances. Some important members are: Formatters, MessageHandlers, and Routes.</td>
</tr>
<tr>
<td>HttpControllerDispatcher</td>
<td>Dispatches an incoming HttpRequestMessage to an IHttpController implementation for processing.</td>
</tr>
<tr>
<td>HttpRequestMessage</td>
<td>Represents a HTTP request message as defined in (IETF, 1999). Some important members are: Content, Headers, Method, RequestUri.</td>
</tr>
<tr>
<td>HttpResponseMessage</td>
<td>Represents a HTTP response message as defined in (IETF, 1999). Some important members are: Content, Headers, StatusCode.</td>
</tr>
<tr>
<td>HttpRouteCollection</td>
<td>A collection of IHttpRoute instances.</td>
</tr>
<tr>
<td>HttpServer</td>
<td>Defines an implementation of an HttpResponseMessage which dispatches an incoming HttpRequestMessage and creates an HttpResponseMessage as a result.</td>
</tr>
<tr>
<td>IHttpRoute</td>
<td>Defines the interface for a route expressing how to map an incoming HttpRequestMessage to a particular controller and action. Some important members are: Constraints, Defaults, RouteTemplate.</td>
</tr>
<tr>
<td>MediaTypeFormatter</td>
<td>Base class to handle serializing and deserializing strongly-typed objects using ObjectContent. Some important members are: SupportedEncodings, SupportedMediaTypes.</td>
</tr>
</tbody>
</table>

\(^{15}\) [http://www.microsoft.com](http://www.microsoft.com)

\(^{16}\) [http://www.odata.org/](http://www.odata.org/)
It is important to mention that as the ASP.NET Web API is a Release Candidate version; thus, these classes may change in later releases. The following figure shows a simplified version of the stack diagram of the ASP.NET Web API.

![Simplified version of the ASP.NET Web API stack diagram.](image)

**Figure 25 – Simplified version of the ASP.NET Web API stack diagram.**

### 5.2.1.1 Ingoing and Outgoing Pipelines

As shown in Figure 25, in ASP.NET Web API all HTTP requests received by the HTTP server pass through a series of message handlers before reaching the controller dispatcher. Each message handler, as discussed in sub-section 4.5.4.2, provides support for different aspects used across various services. According to the diagram presented in Figure 18, ingoing and outgoing pipelines are required. The ASP.NET Web API allows to implement these pipelines by using the Delegation pattern in the handlers, i.e. using the InnerHandler property to specify the next handler in the pipeline, along with the .NET framework asynchronous model as explained next.

Each DelegatingHandler has a SendAsync method that receives two parameters, i.e. a HttpRequestMessage and a CancellationToken, and returns a Task<HttpResponseMessage>. Then, at the end of the SendAsync method, the handler calls the SendAsync method of the base class, i.e. the DelegatingHandler that has the current handler as its InnerHandler. With this model, the logic of the handler for the ingoing pipeline is implemented within the SendAsync method, while the logic of the handler for the outgoing pipeline is implemented using the ContinueWith method of the Task returned from the base handler. The following example illustrates this:
Moreover, it is possible to observe that the ingoing and outgoing pipelines are symmetric; however some aspects are only needed for the request or response, not both. For example, authentication is only needed for the response. Therefore, it is possible to avoid this by not implementing any logic within the SendAsync method or the ContinueWith method. Nevertheless, a problem remains when a particular aspect needs to be executed in different places within these pipelines, e.g. logging needs to be done as soon as the request is received and not just before sending the response, but rather at the middle of the outgoing pipeline. In this case, it is possible to create two handlers for the specific aspect; one implementing logic inside the SendAsync method for the ingoing pipeline and one implementing logic inside the ContinueWith method for the outgoing pipeline.

As mentioned before, handlers have to be registered to be used by the HTTP server. It is important to emphasize that handlers are invoked in the reverse order as when they are added in the MessageHandlers collection of the HttpConfiguration. Table 4 shows the handlers implemented in the Façade layer, while Figure 27 shows how to register these handlers.

<table>
<thead>
<tr>
<th>Handler</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthenticationHanler</td>
<td>Handler to support different authentication mechanisms, including HTTP basic authentication, access key, SWT, JWT, SAML 0.1/0.2.</td>
</tr>
<tr>
<td>CachingHandler</td>
<td>Handler to provide caching support using the caching service of the cloud provider and setting the HTTP headers for caching.</td>
</tr>
<tr>
<td>EncodingHandler</td>
<td>Handler to support compression using gzip and deflate encodings.</td>
</tr>
<tr>
<td>EncryptionHandler</td>
<td>Handler to support security at the message level via encryption.</td>
</tr>
<tr>
<td>LoggingHandler</td>
<td>Handler to log request and response messages using a storage service of the cloud provider.</td>
</tr>
<tr>
<td>ProcessingTypeHandler</td>
<td>Handler to support synchronous and asynchronous processing types for requests.</td>
</tr>
</tbody>
</table>

Table 4 – Handlers implemented in the Façade layer.
5.2.1.2 Controller dispatcher and routing

The next step in the implementation of the Façade layer is related with the Front Controller discussed in sub-section 4.5.4.3. The `HttpControllerDispatcher` is used for this, along with the `HttpRouteCollection` of the `HttpConfiguration`. Each route is composed by the following elements:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Defines a name for the route.</td>
</tr>
<tr>
<td>RouteTemplate</td>
<td>The URI pattern used for the route. Placeholders are specified between {}.</td>
</tr>
<tr>
<td>Defaults</td>
<td>Default values for the route, e.g. default values for the placeholders of the route template.</td>
</tr>
<tr>
<td>Constraints</td>
<td>Expressions that define constraints for the values of the route template or for the verbs accepted by the route.</td>
</tr>
</tbody>
</table>

Table 5 – Elements of a route of the `HttpRouteCollection`.

Figure 28 shows an example of how to define and register routes. In that example, the first route is very simple as it does not have placeholders in the route template. That route maps to a controller named “Orders” and it only receives requests with HTTP verbs GET or POST. The second route has a placeholder “{orderid}” which only accepts numeric values higher than 0. Similarly, this route maps to a controller named “Orders”; however, it accepts request with HTTP verbs GET, PUT or DELETE.
The **HttpControllerDispatcher** uses these routes to select the controller and the specific method within the controller to process the request. Thus, when the **HttpControllerDispatcher** receives a request, it tries to match the URI from the request against the routes registered in the **HttpConfiguration**. A HTTP response with status code 404, i.e. resource not found, is sent to the client if there is no match with any route. Finally, according to the result of the routing process, the **HttpControllerDispatcher** passes the **HttpRequestMessage** to a method of the selected controller to process the request.

### 5.2.1.3 Services

As discussed in sub-section 4.5.4.4, services are composed by a service interface and a service controller. The service interface defines the service contract, i.e. semantics of the incoming and outgoing messages. For this, the service interface can be composed by media type formatters, DTOs and mappers.

In ASP.NET Web API, media type formatters are used to deserialize the body of a request according to the values specified in the **Content-Type** header of a request, or to serialize the body of the response according to the values specified in the **Accept** header of the request. Therefore, it supports a server-driven content negotiation, i.e. the server selects the best representation for a response based on the available representation formats and the values specified by the client (IETF, 1999). In the implementation all resources support JSON and XML; however, DTOs are used in custom media type formatters to define the types supported by the formatter.

On the other hand, the service controller is implemented through the **ApiController** class. Service controllers may have various methods to support different types of resources and operations on these resources. Additionally, it may use different service connector to retrieve data from enterprise systems and provide the required service orchestration. Finally, service controller may contain logic about the
special processing of some resources, e.g. filters, data pagination, synchronous or asynchronous processing, etc.

For example, the service controller in Figure 29 supports the GET HTTP verb for the types of resources listed in Table 6. This service controller uses a channel factory to create a connection with a data service using the Service Bus Relay Service. The services provided by the Service Bus are explained in sub-section 5.2.2.2. The method shown in Figure 30 processes requests for the first type of resource listed in Table 6.

```csharp
namespace Logica.WebAPI.Controllers
{
    public class CategoriesController : ApiController
    {
        // Channel factory to the "categories" service.
        static ChannelFactory<ICategoriesChannel> categoriesChannelFactory;
        static CategoriesController()
        {
            ...
        }
        // GET /categories/
        public HttpResponseMessage GetCategories()
        {
            ...
        }
        // GET /categories/(categoryid)
        public HttpResponseMessage GetCategoryById(int categoryid)
        {
            ...
        }
        // GET /categories/(categoryid)/subcategories
        public HttpResponseMessage GetCategorySubcategories(int categoryid, string subcategories)
        {
            ...
        }
        // GET /categories/(categoryid)/subcategories/(subcategoryid)
        public HttpResponseMessage GetSubcategoryById(int categoryid, string subcategories, int subcategoryid)
        {
            ...
        }
    }
}
```

**Figure 29 – Example of service controller implementation.**

<table>
<thead>
<tr>
<th>Resource URI template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/categories/</td>
<td>List of categories</td>
</tr>
<tr>
<td>/categories/(categoryid)</td>
<td>A particular category</td>
</tr>
<tr>
<td>/categories/(categoryid)/subcategories</td>
<td>Subcategories of a category.</td>
</tr>
<tr>
<td>/categories/(categoryid)/subcategories/(subcategoryid)</td>
<td>A particular subcategory of a category.</td>
</tr>
</tbody>
</table>

**Table 6 – Resources supported by service controller showed in Figure 29.**

```csharp
// GET /categories/
public HttpResponseMessage GetCategories()
{
    CategoryDTO categoryDTO = null;

    using (ICategoriesChannel categoriesChannel = categoriesChannelFactory.CreateChannel())
    {
        CategoryResponseMapper mapper = new CategoryResponseMapper();
        categoryDTO = mapper.MapFromProductCategoryDataList(categoriesChannel.GetProductCategories());
    }

    if (categoryDTO == null)
        throw new HttpResponseMessage(HttpStatusCode.NotFound);

    return Request.CreateResponse<CategoryDTO>(HttpStatusCode.OK, categoryDTO);
}
```

**Figure 30 – Example of a method implementation within a service controller.**
5.2.2 Windows Azure

Besides the reasons to choose Microsoft technologies mentioned above, Windows azure has been selected as the cloud provider for the deployment of this implementation for two reasons. The first reason is related to the integration between Visual Studio and Windows Azure, because this facilitates the testing of the solutions using an emulator of the cloud environment, as well as the deployment and configuration of the solution in the cloud environment. The second reason is related to the conditions for the 90-day free trial subscription. Other cloud providers offer free subscriptions as well; however, according to their conditions, it can be necessary to pay a fee at the end of the month, e.g. in Amazon Web Services\(^\text{17}\) the subscriber has to pay standard rates when free usage expires or if his application use exceeds the free usage tiers.

5.2.2.1 Cloud services and storage

A Windows Azure Cloud Service consists of one or more Web roles and/or Worker roles, each with its own application files and configuration. A Web role is used to deploy a Web application within an Internet Information Services\(^\text{18}\) (IIS) server, while a worker role is used to deploy applications that can run asynchronous, long-running or perpetual tasks independent of user interaction or input. With Windows Azure, it is possible to create several instances of each of these roles as required.

For the implementation, one cloud service has been used to deploy the Web API as well as some additional components of the Façade layer. The cloud service name is “Logica Web API”, its URL is: “http://logicawebapi.cloudapp.net/”, and has been deployed within the “West Europe” data centre of Microsoft. Moreover, a Web role has been used for the Web API, as it is consumed by the mobile enterprise applications via HTTP services. On the other hand, a working role has been used to support asynchronous processing, i.e. when a request is inserted into a queue to be processed later (see 4.2.14). The worker role contains an application that reads and processes requests from a queue (see Figure 31). Finally, another worker role has been used to simulate workflows by processing information on behalf of the users. This worker role contains an application that executes tasks every certain period of time.

Additionally, SQL Database (previously known as SQL Azure) has been used to deploy a database to store logging data, metadata, and other temporal data required by the Façade layer components.

5.2.2.2 Messaging and queuing

The Windows Azure Service Bus has been used to provide support for messaging between the cloud service and the enterprise data services, and to provide support for asynchronous processing of requests. For this purpose, a service namespace has been created with the name “service-bus-logica”; thus, the endpoint of the service is “https://service-bus-logica.servicebus.windows.net/”.

Service Bus Queues has been used within the implementation to support asynchronous processing of requests (see 4.2.14). When a request is processed asynchronously, it is introduced into a queue, so it can be processed later using an application within a worker role as shown in Figure 31. For this purpose, a queue has been created within the service bus namespace created previously. It is possible to create as many queues as needed within the Web API services. Finally, these queue need to be initialized when the Web API application is started within the Web role.

---

\(^{17}\) [http://aws.amazon.com/]

\(^{18}\) [http://www.iis.net/]
On the other hand, Windows Azure Service Bus Relay service has been used to implement messaging communication between the cloud service and the enterprise data services. Within this implementation, the enterprise data services have been deployed on a local machine to demonstrate the communication between cloud services and on-premise services. In order to support this hybrid scenario, i.e. communication between the cloud service and on-premise services, the on-premise service has to start a connection to the relay service through an outbound port, creating a bi-directional socket for communication tied to a particular rendezvous address (Microsoft, 2012). An endpoint address for the enterprise data service in the service bus and a shared secret key are required to establish this connection. With this, the on-premise service registers with the Relay service to listen for messages on the rendezvous address. A TCP connection has been used for the implementation for better performance and throughput. It is worth to mention that it is possible to use HTTP to promote client interoperability; however, in this case the only client is the cloud service.

![Figure 31 – Example of a Working role using a queue from the Service Bus Queue service.](image)

![Figure 32 – Windows Azure Service Bus Relay service. Adapted from (Skonnard & Brown, 2009).](image)
After the bidirectional socket has been created, the cloud service can communicate with the enterprise data services by sending messages to the Service Bus Relay service targeting the rendezvous address. The relay service will then “relay” messages to the on-premise service through the bi-directional socket (see Figure 32). One major advantage of this schema, from the security point of view, is that the enterprise data services do not need any inbound ports open on the firewall in order to be accessed from the cloud service. This same schema can be used in B2B scenarios to access services from partner organizations and exposed them through the Web API to mobile client applications.

5.2.2.3 Access Control and caching

The Windows Azure Access Control service (ACS), the Windows Identity Foundation framework (WIF), and the Thinktecture Identity Model framework (TIM) have been used within the implementation to support authentication and authorization based on the claim-based identity model.

The ACS has been used as a Federation Provider and as an Identity Provider. For this purpose, a service namespace has been created with the name “acs-logica”; thus, the endpoint of the service is “https://acs-logica.accesscontrol.windows.net/”. Additionally, the WIF framework has been used with ASP.NET Web API to have a common programming model for claims. Finally, the TIM framework has been used to validate security tokens within the Web API and handle the authentication and authorization processes. It supports HTTP basic authentication, Access Keys, Simple Web Tokens (SWT), JSON Web Token (JWT), and SAML 1.1/2.0 (Thinktecture, 2012).

When a mobile enterprise application requests a protected resource without a security token, the Web API redirects the application to the ACS. If the application supports various identity providers, the ACS sends a page with the supported identity providers and the user has to select one for the authentication process. Otherwise the ACS directly redirects the application to the login page of the identity provider. Once the user enters his credentials and the authentication is successful, the identity provider issues a security token. The mobile enterprise application presents this security token to the ACS. Finally, the application receives the security token from the ACS and presents it to the Web API.

Similarly, the ACS may act as an identity provider. In this case, when the mobile enterprise application requests a security token, the ACS issues a security token and the application directly presents it to the Web API. It is important to notice that the Web API only receives security tokens issued by the ACS; thus, the Web API can support various identity providers, but only needs a direct trust relation with the ACS.

On the other hand, the Web API uses the AuthenticationHandler from the TIM framework to validate security tokens issued by the ACS. A symmetric key has been generated in the ACS to sign the security tokens. Therefore, the symmetric key is also used by the Web API to verify if the security tokens have been issued by the ACS. If a security token is only used to identify the mobile enterprise application (see 4.2.1), the security token is presented to the Web API using the HTTP authorization header and “CIK” (i.e. Client Identity Key) as the authorization scheme. Moreover, if the security token is used for authentication and authorization, the AuthenticationHandler extracts the claims from the security token and creates a ClaimsPrincipal instance using the WIF framework. The ClaimsPrincipal instance is set as the CurrentPrincipal of the thread used to process the request, and contains the claims extracted from the security token. Finally, the CurrentPrincipal can be used across the Web API to perform authorization tasks during the processing of the request. The following sequence diagram shows this authentication process.
Since HTTP is used as an application protocol between mobile enterprise applications and the Web API of the Façade layer, HTTP caching has been used within the implementation to provide a first level of caching support. The `HttpHelper` class is used within the `CachingHandler` to configure HTTP caching by setting some HTTP headers according to the required caching strategy. For example, an expiration strategy can be implemented by using the `Cache-Control` header within the responses. Similarly, a validation strategy can be implemented by using the `If-None-Match` header within the requests and the `ETag` header within the responses.

```csharp
public HttpResponseMessage GetCategories()
{
    CategoryDTO categoryDTO = null;

    using (IProductCategoriesChannel categoriesChannel = categoriesChannelFactory.CreateChannel())
    {
        CategoryResponseMapper responseMapper = new CategoryResponseMapper();
        categoryDTO = responseMapper.MapFromProductCategoryDataList(categoriesChannel.GetProductCategories());
    }

    if (categoryDTO == null)
        throw new HttpResponseException(HttpHelper.CreateNotFoundResponse());

    HttpResponseMessage response = Request.CreateResponse<CategoryDTO>(HttpStatusCode.OK, categoryDTO);
    response.Headers.CacheControl = HttpHelper.ConfigureCaching(new TimeSpan(0, 5, 0));

    return response;
}
```

However, it is also necessary to provide caching support between the Façade layer and the enterprise data services in order to reduce the number of messages required between the two. Therefore, the Windows Azure Caching service has been used to provide additional caching support. The `CachingHandler`, using a `CachingHelper` class (see Figure 35), provides caching support for GET HTTP requests (see Figure 36). For this purpose, caching has been enabled for the Web Role that hosts the Web API. Additionally, the size of the cache and the duration of an object within the cache
before it expires have been configured. Finally, the cache needs to be initialized when the Web API application is started within the Web role.

```csharp
public class CachingHelper
{
    private static DataCache _cache;

    public static DataCache cache { get { return _cache; } }  

    public static void InitializeCache()
    {
        DataCacheFactory cacheFactory = new DataCacheFactory();
        _cache = cacheFactory.GetCache("default");
    }

    public static void InsertToCache(string key, object value)
    {
    }

    public static void InsertToCache(string key, object value, TimeSpan maxAge)
    {
    }

    public static T GetFromCache<T>(string key)
    {
    }

    public static bool IsInCache(string key)
    {
    }
}
```

*Figure 35 – CachingHelper class using the Windows Azure Caching service.*

```csharp
string cacheKey;

protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request,
    CancellationToken cancellationToken)
{
    if (request.Method == HttpMethod.Get)
    {
        cacheKey = GenerateCacheKey(request);

        if (CachingHelper.IsInCache(cacheKey))
            RespondFromCache(request);

        return base.SendAsync(request, cancellationToken).
            ContinueWith<HttpResponseMessage>((responseToCompleteTask) =>
            {
                var response = responseToCompleteTask.Result;

                WriteToCache(response);
                return response;
            });
    }

    return base.SendAsync(request, cancellationToken);
}

private string GenerateCacheKey(HttpRequestMessage request)
{

private Task<HttpResponseMessage> RespondFromCache(HttpRequestMessage request)
{

private void WriteToCache(HttpResponseMessage response)
{
```

*Figure 36 – CachingHandler using the Windows Azure Caching service.*

5.3 Implementation results and observations

Table 7 briefly describes the implemented requirements of the reference integration architecture:
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Implementation description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSR1. Integration with mobile management solution</td>
<td>Not implemented due to limitations explained in 5.1.</td>
</tr>
<tr>
<td>SSR2. Support different authentication processes</td>
<td>Implemented through the AuthenticationHandler and using a claim-based identity model with Windows Azure Access Control service.</td>
</tr>
<tr>
<td>SSR4. Support for securing enterprise data at message and transport levels.</td>
<td>The support for securing enterprise data at the transport level has been implemented by adding an HTTPS endpoint for the Web role that hosts the Web API. The support for securing enterprise data at the message level and the support for validation, exception management, logging and auditing has been implemented using different HTTP message handlers within the ingoing and outgoing pipelines of the Façade layer.</td>
</tr>
<tr>
<td>SSR5. Support for validation, exception management, logging and auditing.</td>
<td>Implemented using the Windows Azure Service Bus, which supports integration with on-premise and cloud-based applications. On the other hand, by using the claim-based identity model, it is possible to access applications and services from external organizations after establishing a trust relation.</td>
</tr>
<tr>
<td>IIR1. Agnostic to any mobile platform. IIR2. Support diverse types on mobile enterprise applications</td>
<td>Since the Façade layer exposed a Web API, it can be accessed by mobile enterprise applications independently of the mobile OS or type of mobile client. The next chapter explains the implementation of some mobile clients.</td>
</tr>
<tr>
<td>IIR3. Provide integration with diverse types of enterprise systems.</td>
<td>Implemented using the Windows Azure Service Bus, which supports integration with on-premise and cloud-based applications. On the other hand, by using the claim-based identity model, it is possible to access applications and services from external organizations after establishing a trust relation.</td>
</tr>
<tr>
<td>IIR4. Reduce dependencies between mobile enterprise applications and enterprise systems.</td>
<td>Implemented using the Façade layer between the enterprise mobile applications and the enterprise systems. Moreover, it is possible for mobile enterprise applications to continue working even if required enterprise systems are down, by supporting asynchronous processing of requests using the Windows Azure Queue service and the ProcessingTypeHandler.</td>
</tr>
<tr>
<td>IIR5. Provide integration with mobile middleware solutions to support advanced scenarios</td>
<td>Not implemented due to limitations explained in 5.1.</td>
</tr>
<tr>
<td>PS1. Support different caching strategies and various levels of caching</td>
<td>Implemented through the CachingHandler, providing support for expiration and validation strategies. Caching between mobile enterprise applications and the Façade layer is implemented through HTTP caching, while caching between the Façade layer and the enterprise system is implemented using Windows Azure Caching service.</td>
</tr>
<tr>
<td>PS2. Provide mechanisms for content and processing negotiation</td>
<td>Server-driven content negotiation is implemented using the ASP.NET Web API content negotiation mechanisms, i.e. using content negotiation through HTTP Accept and Content-Type headers and media type formatters. Processing negotiation is implemented using the ProcessingTypeHandler and a custom HTTP header.</td>
</tr>
<tr>
<td>PS3. Provide monitoring support</td>
<td>Performance monitoring has been implemented using Windows Azure Monitoring service. Usage monitoring has been implemented via HTTP message handlers; however, no application has been implemented to display the monitoring data collected by the handler.</td>
</tr>
</tbody>
</table>

Table 7 – Summary of implemented requirements of the reference integration architecture.
Chapter 6  Integration Architecture Validation

This chapter describes the proof of concept used for the validation of the reference integration architecture implementation. It starts by describing a fictitious scenario used as a reference to validate the implementation of the requirements for the reference integration architecture. The goal of the fictitious scenario is to illustrate interactions between common types of mobile enterprise applications and enterprise systems. Then it explains the frameworks and tools used for the implementation of the mobile enterprise applications and enterprise data services, and it presents some examples. The limitations presented in sub-section 5.1 apply here as well. Finally, it discusses the results and observations related to the implementation of the mobile enterprise applications and enterprise data services.

6.1 Fictitious scenario

In the fictitious scenario, a fictitious company sells different types of products through online applications. The company has various enterprise systems deployed in the data centre of its main office. The company has implemented a service-oriented architecture using Microsoft technologies and at the moment, its enterprise systems can be accessed through Windows Communications Foundations (WCF) services. For this scenario, the following five services have been used:

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>The service provides access to information about customers.</td>
</tr>
<tr>
<td>Employee</td>
<td>The service provides access to information about employees.</td>
</tr>
<tr>
<td>Order</td>
<td>The service provides access to information about orders. It also permits to create orders, modify an order or update the status of an order.</td>
</tr>
<tr>
<td>Product</td>
<td>The service provides access to information about the categories for products and the products.</td>
</tr>
<tr>
<td>Shopping</td>
<td>The service provides access to information about shopping carts used on e-commerce applications. It also permits to create shopping carts or modify existing ones.</td>
</tr>
</tbody>
</table>

Table 8 – Enterprise data services used in the fictitious scenario.

The company has an online store Web application that consumes some of these services. The Web application allows customers to browse the catalogue of products and order products. Now the company is planning to expand sales by enabling mobile access to its online store via a Web-based mobile application. As a starting point, the company wants to build a Web-based mobile application, as it would be available through any mobile device and mobile OSs. However, they want to see what kind of devices and mobile OSs are more used by their clients to possibly build a native mobile application to provide a better user experience to their customers.

On the other hand, the company wants to provide a mobile application to its sales representatives, so they can have online access to information about customers and orders during their visits to important customers. For this phase, the company has decided to buy Windows phones for its sales workforce to accelerate the implementation of the project; however, they are planning to expand the use of mobile enterprise applications across other areas of the company so they are planning to support other mobile platforms in the near future.

For this purpose, the company has decided to create a solution based on the reference integration architecture presented in Chapter 4. Therefore, the company has implemented the reference integration architecture as described in Chapter 5. The following figure shows an overview of the implementation of the proof of concept based on the fictitious scenario:
6.2 Frameworks and tools

The implementation of the proof of concept is mainly based on Microsoft technologies, due to the same reasons explained in sub-section 5.2. Some of the frameworks and tools used for the implementation of the proof of concept are listed below:

- .NET Framework 4.0
- ASP.NET MVC 4.0
- Enterprise Library 5.0 – May 2011
- Fiddler Web Debugger
- jQuery Mobile
- SQL Server 2012
- SQL Server Management Studio
- Visual Studio 2010
- Windows Identity Foundation
- Windows Identity Foundation SDK
- Windows Communication Foundation
- Windows Phone SDK 7.1

6.2.1 Enterprise Data Services

As explained in sub-section 5.1, the implementation does not include real enterprise systems; thus, the enterprise data services contain business logic and data access logic. However, these services have a
layered internal architecture, as shown in Figure 38, and consume data from a relational database deployed in SQL Server 2012.

![Diagram of layered internal architecture](image)

*Figure 38 – Internal logic architecture for enterprise data services.*

The data access layer is composed by data access components and data access helpers. The data access components abstract the logic required to interact with database objects and uses the Data Access Application Block from the Enterprise Library 5.0 as a data access helper library to connect with the database. Figure 39 shows an example of the interface defined for a data access component.

```csharp
namespace Logica.DAL.DataAccess
{
    public interface IOrderDAL : IPagination
    {
        int OrderId { get; set; }
        int CustomerId { get; set; }
        int EmployeeId { get; set; }
        OrderStatus OrderStatus { get; set; }

        Order GetOrderById();
        List<Order> GetOrders();
        Order AddOrder(Order order);
        void UpdateOrder(Order Order);
        void AssignOrder(Order Order);
    }
}
```

*Figure 39 – Example of the interface for a data access component.*

The business domain layer is composed by a domain model and domain components. The domain model defines business entities and their relations, while the domain components contain the business
logic around the domain model and use the data access components to populate the domain model or to persist data in the database. Figure 40 shows the members of a domain component.

```csharp
namespace logica.BLS.DomainComponent
{
    public class OrderManager
    {
        private IOrderDAL _orderDAL;
        public OrderManager(IOrderDAL orderDAL)
        {
            _orderDAL = orderDAL;
        }
        public Order GetOrderById(int orderId)
        {
            return _orderDAL.GetOrderById(orderId);
        }
        public IList<Order> GetOrders(int rowCount, int pageNumber)
        {
            return _orderDAL.GetOrders(rowCount, pageNumber);
        }
        public IList<Order> GetOrdersByStatus(OrderStatus orderStatus, int rowCount, int pageNumber)
        {
            return _orderDAL.GetOrdersByStatus(orderStatus, rowCount, pageNumber);
        }
        public IList<Order> GetCustomerOrders(int customerId, OrderStatus orderStatus, int rowCount, int pageNumber)
        {
            return _orderDAL.GetCustomerOrders(customerId, orderStatus, rowCount, pageNumber);
        }
        public IList<Order> GetEmployeeAssignedOrders(int employeeId, OrderStatus orderStatus, int rowCount, int pageNumber)
        {
            return _orderDAL.GetEmployeeAssignedOrders(employeeId, orderStatus, rowCount, pageNumber);
        }
        public Order AddOrder(Order order)
        {
            return _orderDAL.AddOrder(order);
        }
        public void ApproveOrders()
        {
            _orderDAL.ApproveOrders();
        }
        public void UpdateOrderStatus(int orderId, OrderStatus newOrderStatus)
        {
            _orderDAL.UpdateOrderStatus(orderId, newOrderStatus);
        }
        public void CancelOrder(int orderId)
        {
            _orderDAL.CancelOrder(orderId);
        }
        public void AssignOrders(List<Employee> employees)
        {
            _orderDAL.AssignOrders(employees);
        }
    }
}
```

*Figure 40 – Example of the member of a domain component.*

Finally, the business service layer is composed by a service model and service contracts. It has been implemented using the Windows Communication Foundation framework. The service model contains the actual implementation of the services. The service contract defines the operations exposed by the service and the data contract for the service, i.e. the description of the data to be exchanged. Finally, as these enterprise data services have to be consumed by the Façade layer through the Service Bus Relay service, they have to implement the `IClientChannel` interface as part of the service contract. Figure 41 shows an example of a service contract. Likewise, an endpoint for the service in the service bus has to be included in the configuration file along with the shared secret key used to communicate using the TCP relay message binding.

```csharp
namespace logica.BLS.ServiceContract.EmployeeService
{
    [DataContract]
    public class EmployeeData
    {
    }

    [ServiceContract(Namespace = "http://logica.BLS.ServiceContract.EmployeeService")]
    public interface IEmployeeContract
    {
        [OperationContract]
        EmployeeData GetEmployeeById(int employeeId);

        [OperationContract]
        IList<EmployeeData> GetEmployees(int rowCount = -1, int pageNumber = -1); 
    }

    public interface IEmployeeChannel : IEmployeeContract, IClientChannel
    {
    }
}
```

*Figure 41 – Example a service contract.*
6.2.2  Web API

As explained in sub-section 6.1, the implementation of the fictitious scenario uses the implementation of the reference integration architecture described in Chapter 5. Therefore, a Web API has been defined to be used by the mobile enterprise applications to access the functionality exposed by the enterprise data services. As discussed in sub-section 4.5.4.4, Web APIs should follow an outside-in or consumer-driven design. Thus, the API should reflect the use-cases required by the various client applications. This provides a level of indirection that enables internal entities to evolve or even be replaced, while minimizing impact on clients. The Web API design has been based on the design process described in (Riva & Laitkorpi, 2007). The following figure shows the resource model for the Web API.

![Resource model for the Web API](image)

Based on the previous resource model, the following API structure has been defined:

<table>
<thead>
<tr>
<th>URI template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/categories</td>
<td>Categories of products</td>
</tr>
<tr>
<td>/categories/{id}</td>
<td>A particular category of products</td>
</tr>
<tr>
<td>/categories/{id}/products</td>
<td>Products within a category of products</td>
</tr>
<tr>
<td>/categories/{id}/subcategories</td>
<td>Subcategories of products within a category</td>
</tr>
<tr>
<td>/categories/{id}/subcategories/{id}</td>
<td>A particular subcategory of products</td>
</tr>
<tr>
<td>/categories/{id}/subcategories/{id}/products</td>
<td>Products within a particular subcategory of products</td>
</tr>
<tr>
<td>/products</td>
<td>Products</td>
</tr>
</tbody>
</table>

Figure 42 – Resource model for the Web API.
Table 9 – API structure for the Web API of the fictitious scenario.

<table>
<thead>
<tr>
<th>URI template</th>
<th>HTTP verb</th>
<th>Supported media types</th>
</tr>
</thead>
<tbody>
<tr>
<td>/categories</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/categories/{id}</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/categories/{id}/products</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/categories/{id}/subcategories</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/categories/{id}/subcategories/{id}/products</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/products</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/products/{id}</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/products/{id}/photos/{size}</td>
<td>GET</td>
<td>image/gif, image/jpg, image/png</td>
</tr>
<tr>
<td>/shoppingcarts</td>
<td>GET, POST</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/shoppingcarts/{id}</td>
<td>GET, PUT</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/orders</td>
<td>GET, POST</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/orders/{id}</td>
<td>GET, PUT, DELETE</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/customers</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/customers/{id}</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/customers/{id}/orders</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/customers/{id}/shoppingcart</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/employees</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/employees/{id}</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
<tr>
<td>/employees/{id}/orders</td>
<td>GET</td>
<td>application/json, application/xml</td>
</tr>
</tbody>
</table>

Table 10 – Supported HTTP verbs and media types for the Web API of the fictitious scenario.

The information from Table 9 and Table 10 has been used to configure the routing table of the Web API. ASP.NET Web API supports JSON and XML media types by default; however, a media type formatter has been created to provide support for the photos of products. Moreover, the following service controllers have been defined:

<table>
<thead>
<tr>
<th>Service controller</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categories</td>
<td>Handles requests to resources under “/categories”</td>
</tr>
<tr>
<td>Customers</td>
<td>Handles requests to resources under “/customers”</td>
</tr>
<tr>
<td>Employees</td>
<td>Handles requests to resources under “/employees”</td>
</tr>
<tr>
<td>Orders</td>
<td>Handles requests to resources under “/orders”</td>
</tr>
<tr>
<td>Products</td>
<td>Handles requests to resources under “/products”</td>
</tr>
<tr>
<td>ShoppingCarts</td>
<td>Handles requests to resources under “/shoppingcarts”</td>
</tr>
</tbody>
</table>

Table 11 – Service controllers for the Web API of the fictitious scenario.
6.2.3 Client Applications

As described in the fictitious scenario presented in sub-section 6.1, the company has an online store implemented as a Web application. In addition, the company wants to implement a Web-based B2C mobile application to enable mobile access to the online store, and a native B2E mobile application to support sales activities of sales representatives. On the other hand, as discussed in sub-section 4.5.4.5, mobile clients can implement the Service Gateway pattern to manage some common tasks when consuming services from the Web API. Therefore, a service gateway has been implemented for the three applications as each one is implemented in a different platform.

The Web application has been implemented using the ASP.NET MVC framework; thus, it implements the Model-View-Controller pattern. In this case, the domain model defined for the enterprise data services has been reused as a model. The views have been implemented using the Razor engine. Finally, the controllers use the service gateway to send requests to the Web API and process the responses. If the response is successful, the controller fills the model with the data from it and returns the appropriate view.

The native mobile application has been implemented using the Windows Phone 7.1 SDK, while the Web-based mobile application has been implemented using ASP.NET MVC with HTML5 and jQuery Mobile. Similarly, the domain model defined for the enterprise data services has been reused and a service gateway has been implemented to handle requests and responses to the Web API for both mobile applications. In this case, the mobile applications use JSON as the default media type and compression via gzip.

6.3 Validation results and observations

This sub-section explains in detail how a request from a mobile enterprise application is processed within the implementation of the reference integration architecture, i.e. the complete lifecycle of a request, from its creation until a response is received. In order to avoid too much complexity, only a common scenario is described, viz., when a mobile enterprise application is requesting the categories of products by sending a HTTP GET request to “http://logicawebapi.cloudapp.net/categories”.

As described in 4.5.4.5, a mobile enterprise application only requires a HTTP client to send requests to the Web API exposed by the Façade layer and it may use a service gateway. Therefore, an instance of the service gateway is created before the application is able to send requests. Once a service gateway instance is created, the mobile enterprise application creates a new request using the service gateway instance and sends it to the Web API using a HTTP client.

On the other hand, a HTTP server within the Façade layer is listening for HTTP requests. Once a request is received, the HTTP server passes it to the first handler within the pipeline. As shown in Figure 27, the first handler within the pipeline is the EncryptionHandler, which verifies if the body of the request is encrypted. If so, the handler decrypts it before passing the request to the next handler. The next handler is the LoggingHandler. This handler logs information about the request asynchronously, i.e. the request is passed to the next handler without waiting for the logging process to be completed. The next handler is the AuthenticationHandler. This handler verifies if the request has a security token. If the request does not have a security token, the handler interrupts the request processing and returns a response with the 401 HTTP status code (IETF, 1999); thus, the mobile enterprise application has to request a security token from the Windows Azure Access Control service as described in sub-section 5.2.2.3. If the request contains a security token, the handler validates the token, sets the CurrentPrincipal object with the claims obtained from the token, and passes the request
to the CachingHandler. This handler verifies if the request can be served from the cache in the Windows Azure Caching service. If this is the case, the handler interrupts the request processing and creates a response using the data in the cache; otherwise it passes the request to the EncodingHandler. This handler verifies if the body of the request is encoded and, if so, decodes it before passing the request to the ProcessingTypeHandler. This is the final handler in the pipeline and it verifies if the request should be processed asynchronously or synchronously. In the first case, the request is inserted into a queue from the Windows Azure Queue service, for later processing as described in 5.2.2.2, and a response is sent with 202 HTTP status code, i.e. the request has been accepted but the processing has not been completed (IETF, 1999). If the request has to be processed synchronously, the request is forwarded to the Front Controller.

The Front Controller uses the URI from the request and the routing table information to select a service controller. Additionally, the body of the request is processed and serialized into a DTO using the media type formatters. Finally, the query string parameters and the HTTP verb used in the request are used to select a method within the service controller to process the request. As explained in subsection 5.2.1.3, the methods of a service controller have all the required logic to process a request. In this case the request is assigned to the GetCategories() method of the CategoriesController. This method uses the ChannelFactory component of the Façade layer to create a channel to the “sb://service-bus- logica.servicebus.windows.net/categories” endpoint created in the Windows Azure Service Bus. This channel is used to connect to the Product enterprise data service using the Windows Azure Service Bus Relay service and execute the GetProductCategories() operation.

Internally, the GetProductsCategories operation uses the ProductCatalogue class, which is part of the business layer of the enterprise data services. Similarly, this class uses the ProductDAL component, which is a component of the data access layer and manages the communication with the repository where the categories of products are stored. The ProductDAL component executes the stored procedure “Production.GetProductCategories” to retrieve the categories of products from the SQL Server database and populate a collection with ProductCategory objects defined in the domain model. The collection is returned to the ProductCatalogue and then to the Product service. Finally, the Product service maps the domain model objects according to the service data contract and sends a response using the channel created for the operation.

Once the response is received by the GetCategories() method of the CategoriesController service controller, the channel is closed and a HttpResponseMessage object is created. Then the collection of categories is mapped to a DTO, which is set as the body of the response. Finally, the HttpResponseMessage is returned as a result of the GetCategories() method; thus, the response is serialized using a media type formatter according to the media type requested in the Accept HTTP header and passed again to the pipeline.

In this case, the handlers are executed in reverse order. Therefore, the ProcessingTypeHandler is called first. However, this handler does not implement any logic for the outgoing pipelines; therefore, the response is passed to the EncodingHandler. This handler checks if an encoding was defined in the request by inspecting the Accept-Encoding HTTP header and encodes the response if so before passing it to the next handler. The CachingHandler sets the HTTP header for caching according to the selected caching strategy, then it writes the response in the cache of the Façade layer for future requests and passes it to the AuthenticationHandler. Again, this handler does not implement any logic for the outgoing pipeline so it passes the response to the next handler. As with the request, the LoggingHandler logs information about the response asynchronously and forwards it to the next
handler. Then, the EncryptionHandler verifies if the response needs to be encrypted, in which case encrypts the request before it is sent back to the mobile enterprise application through the HTTP server.

Then, the HTTP client receives the response and passes it to the service gateway. Finally, the service gateway may require to decrypt or decode the response and deserialize it before the returned data can be used by the mobile enterprise application.

Although this is the simpler scenario, it illustrates some of the steps and components required to process requests from mobile enterprise application to consume data or functionality from enterprise systems in a consistent and secure manner, even if these latter are deployed on premise or in the cloud, or if they are managed internally or by an external organization. Moreover, by taking advantage of HTTP and many services from Windows Azure, it is possible to provide scalability and improved performance.
Part III: Conclusions
Chapter 7  Conclusions and final remarks

7.1 Conclusions

As stated in Chapter 1, the goal of this master thesis is to determine how enterprises can use Web technologies to provide seamless integration between mobile enterprise applications and enterprise systems. For this purpose, several research questions have been defined and answered during the different phases of this master thesis. The following paragraphs present a brief summary of the answers for these questions as well as some conclusions.

As discussed in Chapter 2, the main four challenges for enterprise mobility are related to privacy, security, management and integration. Since people is using their mobile devices for personal and professional purposes, it is important to address the associated privacy issues of having personal and enterprise data on the same device as part of the enterprise mobility strategy. Similarly, since security is one of the main factors that prevent companies from implementing mobile solutions, it is important to consider the various aspects of security discussed in sub-section 2.1.2 as part of the enterprise mobility strategy. Likewise, it is important to define an approach for the management of mobile solutions and identify which skills and capabilities are required to support the processes related to enterprise mobility. Finally, it has been observed that enterprises have to extend their integration architectures and integration strategies to achieve a seamless integration with mobile applications.

The different types of mobile enterprise applications, their main characteristics and challenges, the most common approaches, as well as their influence on the mobile integration strategy have been discussed in Chapter 2. Three different types of mobile enterprise applications have been identified, viz. B2B, B2C and B2E mobile applications, as well three common approaches for implementing them, viz. native, hybrid and Web-based mobile clients. This has been very valuable since each type of application has different concerns and each approach has different advantages and disadvantages that may influence the requirements of a particular mobile enterprise solution and the mobile integration strategy. On the other hand, it has been possible to identify some important challenges for these applications associated with the mobile and enterprise environments. From these challenges, one can observe the importance of considering aspects like adaptability, resource constraints, cross-platform support, heterogeneity, security and IT governance during the design and implementation of mobile solutions, as these aspects may influence various trade-off decisions about the mobile integration strategy.

The challenges for mobile enterprise application integration and the integration aspects that have to be considered during the design and implementation of mobile enterprise solutions have been discussed in Chapter 3. Mobile enterprise application integration has challenges related to enterprise application integration, viz. integration at the data, application, process and inter-organizational levels, heterogeneity at each of these levels, constant changes in the business requirements, and fast evolution of information and communication technologies. Additionally, it has challenges related to distributed systems, viz. network reliability, latency, bandwidth, network security, changes in network topology, multiple system administrators, transport costs and network heterogeneity. Therefore, it has been possible to determine the required support for security, integration, interoperability, performance and scalability in mobile enterprise solutions in order to overcome many of these challenges. On the other hand, different approaches as well as the use of Web technologies to enable and support integration between mobile enterprise applications and enterprise systems have been discussed. It has been observed that functional integration is preferred and that it is possible to use Web technologies to extend current implementations of service-oriented architectures in order to provide optimized APIs for the integration of mobile applications.
Finally, it is possible to derive the following conclusions from this work:

- Due to the constant changes in the mobile market and the fast evolution of mobile technologies, companies must have a flexible and extensible mobile integration strategy; therefore, companies should avoid lock-ins situations in mobile solutions.
- Companies have to consider enterprise mobility from a more strategic point of view. Nowadays, many vendors of mobile solutions offer products that promise to solve a lot, if not all, challenges associated with mobile enterprise solutions. However, each enterprise is different and has particular requirements; therefore, a one-size-fits-all approach is not enough. Enterprise should first identify their particular requirements, and understand the challenges associated with the implementation of mobile enterprise solutions according to their current situation, before deciding which solutions are more suitable for them and how those solutions can be integrated within their current enterprise application portfolio.
- Mobile solutions are very complex and even more within an enterprise environment; therefore, trade-off decision must be carefully evaluated. Especially decisions related with performance and security, since these two aspects are particularly conflicting with each other. Consequently, the strategies for implementing mobile enterprise solutions should be flexible enough, so these aspects are balanced according to the specific requirements of a mobile solution.

7.2 Contribution of this work

This work has two main contributions. On one hand, the results of the research phase, i.e. the literature survey and the research within the mobile development teams within Logica, give various important insights on the current challenges in enterprise mobility, mobile enterprise applications and mobile enterprise application integration, as well as on current solutions and approaches to overcome these challenges. Furthermore, the research has allowed to identify common scenarios for mobile enterprise applications. Finally, these results provide an overview of the different infrastructure and integration requirements associated with mobile enterprise solutions.

On the other hand, the presented reference integration architecture encapsulates several architectural and design patterns that have proven to be useful in the domain of mobile and enterprise applications. Moreover, it identifies the different components required to support various important aspects of mobile solutions, as well as their distribution and relations. Furthermore, the reference integration architecture can be used as a baseline for the design and implementation of mobile enterprise solutions. Therefore, it is a valuable tool for enterprise and software architects to identify, evaluate and communicate the requirements for this kind of solutions. Additionally, the implementation of the reference integration architecture, described in chapters 5 and 6, shows how to use it and shows some frameworks and tools that are currently available for this purpose. Finally, the reference integration architecture provides various extensions points in order to be extended according to the particular conditions and requirements of an enterprise.

7.3 Future work

As mentioned earlier in this document, there have been significant improvements in the supporting mobile information and communication technologies in recent years, and these technologies continue evolving in a rapid pace. Similarly, the requirements for mobile solutions are higher and higher and change from time to time. Therefore, enterprises need more supporting tools for designing and implementing integrated mobile enterprise solutions. Furthermore, these tools should be flexible and extensible enough to deal with the dynamism of the mobile and enterprise environments.
This master thesis presents a reference integration architecture for mobile enterprise solutions; however, the reference integration architecture is only a starting point for enterprise architects and software architects when designing and implementing mobile solutions within a company. Due to limitations previously explained, the proposed reference integration architecture should be further developed and tested, including emerging scenarios and new solutions from solutions’ vendors. A list of future work to improve and refine the reference integration architecture is presented next:

- The reference integration architecture should be tested with more complex implementations, i.e. including mobile application development platforms, mobile middleware solutions and mobile management solutions.
- The reference integration architecture should be refined according to different enterprise mobility strategies.
- A comparison between on-premise and cloud deployment for the façade layer should be performed, as well as a deeper analysis of the advantages and disadvantages of each approach.
- More research should be carried out about the design and implementation of Web APIs, including API design and API management strategies. Likewise, a framework to compare and evaluate the different solutions for exposing Web APIs and consuming Web APIs from external organizations should be developed.
References


[Accessed 14 March 2012].


Available at: http://www.davidchappell.com/writing/white_papers.php


Available at: http://www.gartner.com/id=1635614
[Accessed 26 January 2012].

Available at: http://www.gartner.com/id=1997415
[Accessed 09 May 2012].


Krebs, D. & Klein, E., 2012. *Native & HTML5 Mobile Apps: Not an either or, but a where and when*, s.l.: VDC Research.


Available at: https://github.com/thinktecture/Thinktecture.IdentityModel.40
[Accessed 15 July 2012].


Available at: http://www.w3.org/TR/webarch/
[Accessed 26 February 2012].

Available at: http://www.w3.org/TR/ws-arch/
[Accessed 26 January 2012].

Worklight, 2011. Webinar - Native, Web or Hybrid Mobile App Development?. [Online]
[Accessed 27 February 2012].


Available at: http://www.xrgsystems.com/solution.php?page=mobile-applications
[Accessed 27 February 2012].

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ACS</td>
<td>Access Control Service</td>
</tr>
<tr>
<td>B2B</td>
<td>Business-to-Business</td>
</tr>
<tr>
<td>B2C</td>
<td>Business-to-Consumer</td>
</tr>
<tr>
<td>B2E</td>
<td>Business-to-Employees</td>
</tr>
<tr>
<td>BYOD</td>
<td>Bring Your Own Device</td>
</tr>
<tr>
<td>CSS3</td>
<td>Cascading Style Sheets level 3</td>
</tr>
<tr>
<td>EMS</td>
<td>Enterprise Mobility Strategy</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>HTML5</td>
<td>HyperText Markup Language version 5</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IIS</td>
<td>Internet Information Services</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JWT</td>
<td>JSON Web Token</td>
</tr>
<tr>
<td>MAM</td>
<td>Mobile Application Management</td>
</tr>
<tr>
<td>MADP</td>
<td>Mobile Application Development Platform</td>
</tr>
<tr>
<td>MCAP</td>
<td>Mobile Consumer Application Platform</td>
</tr>
<tr>
<td>MDM</td>
<td>Mobile Device Management</td>
</tr>
<tr>
<td>MEAP</td>
<td>Mobile Enterprise Application Platform</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>REST</td>
<td>REpresentational State Transfer</td>
</tr>
<tr>
<td>ROA</td>
<td>Resource-Oriented Architecture</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>STS</td>
<td>Security Token Service</td>
</tr>
<tr>
<td>SWT</td>
<td>Simple Web Token</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TIM</td>
<td>Thinktecture Identity Model</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>WDSL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WIF</td>
<td>Windows Identity Foundation</td>
</tr>
<tr>
<td>WOA</td>
<td>Web-Oriented Architecture</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
Glossary

**Application Programming Interface (API)**
The external interface of a software platform, such as an operating system, that is used by systems or applications built on top of it (Buschmann, et al., 1996).

**Architectural Style**
An architectural style is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style (Fielding, 2000).

**Bring Your Own Device (BYOD)**
Refers to employees taking their own personal device to work, whether laptop, smartphone or tablet, in order to interface to the corporate network (PC Magazine, 2012).

**Claims-based identity model**
A way to write applications so that the establishment of user identity is external to the application itself. The environment provides all required user information in a secure manner (Baier, et al., 2011).

**Content Negotiation**
The process of selecting the best representation for a given response when there are multiple representations available (IETF, 1999).

**Design Pattern**
A design pattern provides a scheme for refining elements of a software system or the relationships between them. It describes a commonly recurring structure of interacting roles that solves a general design problem within a particular context (Buschmann, et al., 2007).

**Enterprise Mobility**
The ability of an enterprise to connect to people and control assets from any location. Technologies that support enterprise mobility include wireless networks, mobile applications, middleware, devices, and security and management software (Mccammon, 2011).

**Enterprise Systems**
Software products designed to integrate computer systems that run all phases of an enterprise’s operations to facilitate cooperation and coordination of work across the enterprise. The intent is to integrate core business processes (e.g., sales, accounting, finance, human resources, inventory and manufacturing). The ideal enterprise system could control all major business processes in real time via a single software architecture on a client/server platform. Enterprise software is expanding its scope to link the enterprise with suppliers, business partners and customers (Gartner, 2012).

**Federated Identity**
A mechanism for authenticating a system’s users based on trust relationships that distribute the responsibility for authentication to a claims provider that is outside of the current security realm (Baier, et al., 2011).

**HTTP**
Is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as
name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred (IETF, 1999).

**Identity provider**
An organization issuing claims in security tokens. For example, a credit card provider organization might issue a claim in a security token that enables payment if the application requires that information to complete an authorized transaction (Baier, et al., 2011).

**Integration Architecture**
The technology architecture of two or more connected applications or systems including whatever technologies, resources, or extensions were added to enable their integration. Many integration architectures include middleware platforms and associated adapter or bridging extensions (Erl, 2009).

**IT Governance**
The distribution of IT decision-making rights and responsibilities among enterprise stakeholders, and the procedures and mechanisms for making and monitoring strategic decisions regarding IT (Peterson, 2004).

**Mobile Application**
A software application that runs in a smartphone, tablet or other portable device (PC Magazine, 2012).

**Mobile Middleware**
Middleware designed to address specific challenges faced by mobile applications running on wireless links that may be slow, intermittent or have high latency. Mobile middleware performs functions such as protocol optimization, data synchronization and data compression (Gartner, 2012).

**Reference Architecture**
A reference architecture is the collection of concepts and patterns of structures and texture that allow the systems conforming to the same reference architecture to interoperate and to be managed by the same tools and procedures (Kruchten, 2000).

**Representation**
Data that encodes information about resource state (W3C, 2004).

**Resource**
Anything that might be identified by a URI (W3C, 2004).

**Service**
An abstract resource that represents a capability of performing tasks that represents a coherent functionality from the point of view of provider entities and requester entities. To be used, a service must be realized by a concrete provider agent.” (W3C, 2004).

**Security Token Service**
A claims provider implemented as a Web service that issues security tokens (Baier, et al., 2011).
**Solution**
A solution is an implementation of people, processes, information and technologies in a distinct system to support a set of business or technical capabilities that solve one or more business problems (Gartner, 2012).

**URI template**
URI Templates are expressions that define how requested URIs should be parsed. Templates divide URIs into multiple URI segments and query strings. A segment is an alphanumeric value that occurs between delimiters such as forward slashes and semicolons (Daigneau, 2011).

**Web API**
An application programming interface (API) of a service implemented using Web technologies.

**Web service**
A Web service is a software system designed to support interoperable machine-to-machine interaction over the network. It has an interface described in a machine processable format (specially WDSL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards (W3C, 2004).