Towards Robustness and Self-Organization of ESB-based Solutions using Service Life-cycle Management

Arnaldo António Pinto Pereira

Final project report submitted to the School of Technology and Management of Bragança to obtain the degree of Master in Information Systems
Relatório final do trabalho de projeto apresentado à Escola Superior de Tecnologia e Gestão de Bragança para obtenção do grau de Mestre em Sistemas de Informação

Supervised by/Orientado por
Prof. Doutor Paulo Leitão

Bragança
2014

Towards Robustness and Self-Organization of ESB-based Solutions using Service Life-cycle Management

Arnaldo António Pinto Pereira
URL: www.ipb.pt/~arnaldo
E-mail: arnaldo@ipb.pt

Final project report submitted to the School of Technology and Management of Bragança to obtain the degree of Master in Information Systems
Relatório final do trabalho de projeto apresentado à Escola Superior de Tecnologia e Gestão de Bragança para obtenção do grau de Mestre em Sistemas de Informação

Supervised by/Orientado por
Prof. Doutor Paulo Leitão

Polytechnic Institute of Bragança
Campus de Santa Apolónia
5301-854 BRAGANÇA, PORTUGAL

December 2014

To my family, my girlfriend Prazeres and friends.

Acknowledgements

This work was supported by the European Union FP7 Programme under the ARUM project No. 314056.

I would like to thank to the supervisor of this project, Professor Paulo Leitão, for the valuable guidance and advice, support and motivation. Completion of this work would not have been possible without his help.

My gratitude also goes to my colleagues, Adriano, Filipe, Jonas, José and Nelson.

Finally a word of recognition to the School of Technology and Management of Bragança for the availability and support given.

Abstract

Enterprise Service Bus (ESB) is a middleware infra-structure that provides a way to integrate loosely-coupled heterogeneous software applications based on the Service Oriented Architecture principles. The life-cycle management of services in such highly changing environments is a critical issue for the component's reuse, maintenance and operation.

This work introduces a service life-cycle management module that extends the traditional functionalities with advanced monitoring and data analytics to contribute for the robustness and self-organization of networks of clusters based on ESB platforms. The realization of this module was embedded in the JBoss ESB, considering a sniffer mechanism to collect relevant details of the service messages crossing the bus. A Liferay portal was created to display information related to the services' health.

Keywords: Service Oriented Architecture, Enterprise Service Bus, Life-Cycle Management, Robustness, Self-organization

Resumo

Um Barramento de Serviços (BS) é um *middleware* que oferece uma infraestrutura que permite a integração de aplicações heterogéneas, com base nos princípios das Arquiteturas Orientadas aos Serviços. A gestão do ciclo de vida dos serviços em tais ambientes altamente dinâmicos é um problema crítico com impacto na operação e manutenção do sistema, bem como na capacidade de reutilização de componentes.

Este trabalho apresenta um módulo de gestão do ciclo de vida de serviços, que acrescenta às funcionalidades tradicionais a monitorização avançada e a análise de dados, no sentido de contribuir para a robustez e auto-organização de redes de agregados computacionais baseados em plataformas BS. O módulo foi integrado no JBoss ESB e utilizou-se um *sniffer* para recolher os detalhes das mensagens que se encontram a ser trocadas no barramento. Foi criado um portal Liferay para apresentar a informação relativa à saúde dos serviços.

Palavras-chave: Arquitetura Orientada aos Serviços, Barramento de Serviços, Gestão do Ciclo de Vida de Serviços, Robustez, Auto-organização

Contents

List of Figures

List of Tables

Acronyms

ACL	Agent Communication Language
ADACOR	Adaptive Holonic Control Architecture for Distributed Manufacturing Systems
ARUM	Adaptive Production Management
BS	Barramento de Serviços
CPAL-1.0	Common Public Attribution License Version 1.0
CPS	Cyber-Physical Systems
DF	Directory Facilitator
ESB	Enterprise Service Bus
FIPA	Foundation for Intelligent Physical Agents
FP7	7th Framework Programme for Research and Technological Development
GUI	Graphical User Interface
ICT	Information Communication Technology
iESB	intelligent Enterprise Service Bus
IoT	Internet of Things
IT	Information Technology
JMS	Java Message Service
KPI	Key Performance Indicator
LCMM	Service Life-cycle Management Module
LGPLv2.1	GNU Lesser General Public License, version 2.1

MAS	Multi-Agent System
MES	Manufacturing Execution System
OMG	Object Management Group
QoS	Quality of Service
REST	REpresentational State Transfer
SAWSDL	Semantic Annotations for WSDL and XML Schema
SOA	Service Oriented Architecture
UDDI	Universal Description, Discovery and Integration
UI	User Interface
W3C	World Wide Web Consortium
WS-BPEL	Web Services Business Process Execution Language
WS-CDL	Services Choreography Description Language
WSDL	Web Service Description Language
XML	eXtended Markup Language

Chapter 1

Introduction

The conceptualization of Internet of Things (IoT) paradigm and the implementation of computational distributed systems reinforce the importance of the integration of heterogeneous software applications across the enterprise Information Technology (IT) infrastructures. In fact, according to a prediction report from Gartner on Application Integration [Gartner, 2012], by 2016, midsize to large companies will spend 33% more on application integration than in 2013, and by 2018, more than 50% of the cost of implementing 90% of new large systems will be spent on integration.

The advent of Service Oriented Architecture (SOA) [Erl, 2005] as a software paradigm for distributed systems to integrate the enterprise IT infrastructures brought the concept of Enterprise Service Bus (ESB). An ESB is a software architecture model used for designing and implementing the communication between interacting software applications in a SOA environment. It is based on the idea to have a standard and structured middleware that offers a way to connect and integrate loosely-coupled heterogeneous software components, named services, reducing the complexity of application interfaces. In 2003, Gartner predicted that the majority of large enterprises will have an ESB running to integrate their IT infrastructures in the near future [Gartner, 2002]. Several ESB products are available in the form of commercial and open source products, namely Oracle Service Bus, Mule ESB, Red Hat JBoss Enterprise Application, TIBCO ActiveMatrix BusinessWorks and JBoss ESB.

The ESB solutions provide a distributed, modular and pluggable architecture, supporting dynamic routing and mediation of services' discovery, request and execution. The main benefits of using an ESB platform is the increased flexibility and scalability, the interoperability transparency and the existence of configuration rather integration coding. In fact, large-scale distributed systems can benefit from an ESB middleware acting as a broker between the nu-

merous heterogeneous service providers/requesters, avoiding a potentially huge number of point to point connections. However, the increased overhead and the possible slower communication speed are the main disadvantages of ESB solutions.

1.1. The Problem

One of the main functionalities of an ESB is to monitor and control the routing of messages exchanged between services. The life-cycle management of deployed services, e.g., including functions of monitoring and data analytics, is a crucial issue for the component reusing, maintenance and monitoring [Wang et al., 2012], and in the context of ESBs, contributes to increase the system robustness, reliability and fault-tolerance. In fact, the possibility to monitor the performance of registered services and analyse the evolution of their behaviour, allows to detect in advance the possible degradation or risk propagation, being possible to generate warnings to implement proper corrective actions to mitigate the problem.

Currently, and related to the life-cycle management, the ESB platforms only provide basic functions associated to the service registry and completely misses these kind of advanced functionalities, leading to the need to have a life-cycle management functionality embedded in the ESB that provides monitoring and data analytics of the registered services.

1.2. Objectives and Contributions

The objective of this work is to develop a life-cycle management module that may be embedded in the traditional ESBs to provide advanced monitoring and data analytics of the registered and deployed services, contributing to achieve more robust, reliable and fault-tolerant distributed SOA-based systems. Additionally, this functionality will also play an important role in the self-organization of the network of software applications organized as clusters of ESBs, in a dynamic and on-the-fly manner.

The ultimate goal of this work is the integration of this module in the final software deliverable of ARUM (Adaptive Production Management)[1] project that will be used in the final review meeting that will be held at the Airbus industrial facility in Hamburg.

[1] http://www.arum-project.eu/

1.3. Document Organization

This document is organized into five chapters, the first of which consists of this introduction where the problem statement and objectives were defined. The rest of the work is organized as follows:

Chapter 2 shows an overview of the state of the art about SOA and ESB, pointing to the more technical aspects. Besides the description of SOA principles and the ESBs' characteristics, it is presented a closer look in Service Oriented Multi-agent Systems and Web services.

Chapter 3 discusses the architecture of the intelligent enterprise service bus (iESB) developed under the ARUM project, and presents the specification of the life-cycle management module as part of the iESB. The contribution of this module to achieve robustness and self-organization is also detailed, as well as technical details related to the implementation of experimental prototype.

Chapter 4 discusses how the module contributes to achieve robustness and self-organization, by presenting two experimental scenarios and analysing the achieved results.

At last, Chapter 5 rounds up the work with the conclusions and points out some future work.

Chapter 2

Service Oriented Architecture and Enterprise Service Bus

IT infrastructure is suitable for the servicification by applying SOA approaches. Several success examples can be reported related to the application of SOA principles, e.g., Amazon.com[2] has gone from being an online retailer to be a dominant e-commerce platform by exposing services to their partners using SOA [Harris, 2007]. In 2007, the use of SOA concepts allowed the British Telecom[3] to close down 800 systems [Lawson, 2011]. The Federal Aviation Administration (FAA)[4] used SOA and cloud computing for its National Airspace System, allowing the implementation of the next generation of air traffic management systems [Hritz, 2012].

Beyond business applications, due to the promised agility and flexibility benefits, embedded devices offering services in the context of Cyber-Physical Systems (CPS), namely in automation and manufacturing environments, are no novelty. In fact, as pointed out by [Mendes, 2011], SOA fits well with collaborative automation, "*addressing distributed, modular and reconfigurable automation systems whose behavior is regulated by the coordination of services*".

This chapter includes a comprehensive summary of the state of the art related with SOA and ESB. The option is to visit the central concepts without delays with historical considerations, focusing the technical aspects.

[2] http://www.amazon.com/

[3] http://www.bt.com/

[4] http://www.faa.gov/

2.1. SOA Principles

Browse by different contributions related to SOA can be a challenging job. Following the popularity associated with SOA, there is an explosion of individual and institutional views that try to encompass all concepts considered important. This is particularly evident observing the contributions from the open standards organizations like OASIS[5], the Object Management Group (OMG)[6], The Open Group[7] [Kreger et al., 2009] and the World Wide Web Consortium (W3C)[8]. Next lines present a high level overview of the SOA concepts, using as guidance the contributes of those organizations.

The best way to frame the concept of SOA is starting to present the different definitions established by the aforementioned organizations:

- "*Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.*" [OASIS, 2006]
- "*Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation. Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services.*" [The Open Group, 2011a]
- SOA is "*An architectural paradigm for defining how people, organizations and systems provide and use services to achieve results.*" [OMG, 2012]
- SOA is "*A set of components which can be invoked, and whose interface descriptions can be published and discovered.*" [W3C, 2004a]

The same approach can be used for the concept of "service", listing the different definitions:

- Service is "*The means by which the needs of a consumer are brought together with the capabilities of a provider.*" [OASIS, 2006]

[5] https://www.oasis-open.org/

[6] http://www.omg.org/

[7] http://www.opengroup.org/

[8] http://www.w3.org/

- *A service is a logical representation of a repeatable business activity that has a specified outcome*" (…) "*and: Is self-contained; May be composed of other services; Is a* "*black box*" *to consumers of the service*". [The Open Group, 2011a]
- "Service *is defined as a resource that enables access to one or more capabilities. Here, the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. This access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. A service is provided by an entity – called the* provider *– for use by others. The eventual* consumers *of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider.*" [OMG, 2012]
- "*A service is an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requesters entities. To be used, a service must be realized by a concrete provider agent.*" [W3C, 2004a]

After processing the central documents of open standards organizations, it is clear that SOA is a way to build distributed systems where the central piece is the concept of service that service providers offers to service consumers. A service is the implementation of some business logic made accessible through a well-defined interface (contract), hiding the implementation details.

Service consumers and providers must find themselves visible. Preconditions to visibility are awareness, willingness (predisposition to interact) and reachability (participants must be able to communicate with each other). Awareness prescribes that the service consumer must have information that would lead to know of the service provider's existence [OASIS, 2006]. Using discovery mechanisms, service consumers find the services they need, and interact directly with those services. These discovery mechanisms are not specified at this abstract level, but two main approaches can be considered: using a registry or considering a peer-to-peer solution [W3C, 2004b].

Services with more complexity can be composed by other services (in this context classified as atomic services). Two main types of composition are often distinguished: orchestration, in which one of the services schedules and directs the others [W3C, 2004a] and choreog-

raphy, in which the composed services interact and cooperate without the aid of a directing service in a peer-to-peer way [W3C, 2005].

2.2. Service Oriented Multi-agent Systems

Notwithstanding the several interpretations, a possible definition for agent is [Leitão, 2009]: "*An autonomous component that represents physical or logical objects in the system, capable to act in order to achieve its goals, and being able to interact with other agents, when it doesn't possess knowledge and skills to reach alone its objectives*". An agent exhibits autonomy and cooperation, and may have reasoning and learning capabilities. For instance, in the manufacturing domain, an agent can represent a physical resource, such as a machine, a robot or a pallet, or a logical object, such as a scheduler or an order. A Multi-Agent System (MAS) can be defined as a set of agents that represent the objects of a system, capable of interacting, in order to achieve their individual goals, when they don't have enough knowledge and/or skills to achieve individually their objectives [Leitão, 2009] (note that each agent has a partial view of the system and none agent has a complete view of the system). These systems have the capability to respond promptly and correctly to change, and differ from the conventional approaches due to their inherent capabilities to adapt to emergence without external intervention [Wooldridge, 2002].

The service-oriented principles can be integrated with MAS to enhance some functionality and to overcome some of its limitations, namely in terms of interoperability and IT-vertical integration. Indeed agents are already present in standard documents of SOA (e.g., see [OASIS, 2006]) and, at the same time, services are already part of the agents' specification [FIPA, 2002a]. In spite of being based on the same concept of provide a distributed approach to the system, MAS and SOA present some important differences, namely in terms of computational requirements and interoperability (see [Ribeiro et al., 2008] for a deeply study). These differences (presented in Table 1) highlight the complementary aspects of the two paradigms, suggesting the benefits of combining them.

MAS	SOA
Well established methods to describe the behaviour of an agent	Focus is on detailing the public interface rather than describing execution details

Agents denote social ability regulated by internal or environmental rules	Social ability is not defined for SOA
Most implementations are optimized for LAN use	Supported by Web related technologies and can seamlessly run on the Internet
Reactive to changes in the environment	Reconfiguration often requires reprogramming
Interoperability heavily dependent on compliance with FIPA-like standards	Interoperability assured by the use of general purpose web technologies
Heavy computational requirements	High performance without interoperability constraints

Table 1: Differences between MAS and SOA [Ribeiro et al., 2008].

Traditionally, the combination of MAS and SOA paradigms can be performed in different ways, as illustrated in Figure 1 [Mendes et al., 2009].

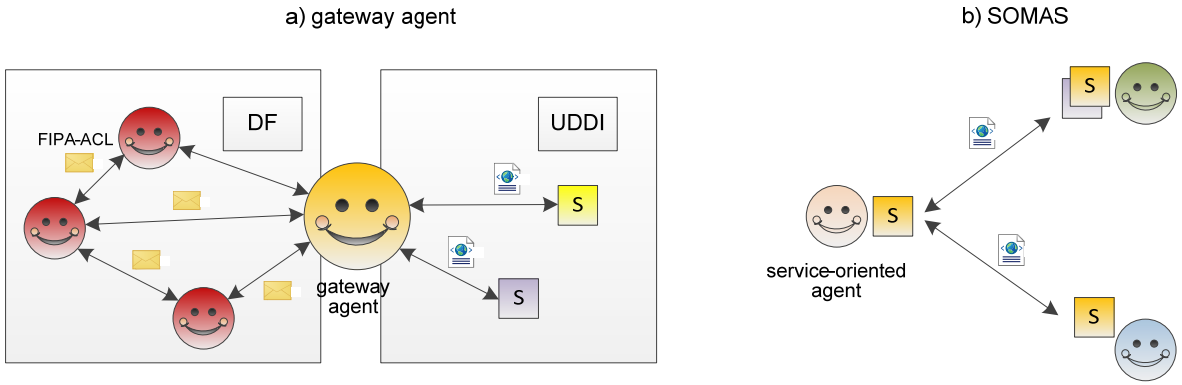

Figure 1: Common approaches for integrating SOA and MAS (adapted from [Mendes et al., 2009]).

The first traditional option, illustrated in Figure 1.a), considers gateways to translate the semantics from the agents world to the services world. This involves translate entries on Directory Facilitator (DF) to/from Universal Description, Discovery, and Integration (UDDI) registry entries. Equally translated are agents' skills to/from Web Service Description Language (WSDL) instances. Finally, message translation is offered between Agent Communication Language (ACL) messages and SOAP[9] messages [Greenwood et al., 2007]. However,

[9] Since version 1.2, is not an acronym. SOAP was originally an acronym for Simple Object Access Protocol.

using the described approach, the design of truly service-oriented multi-agent systems are far from the real expected potential and benefits. Another option, illustrated in Figure 1.b), was introduced by [Mendes et al., 2009] and is characterized by the use of a set of autonomous agents that use the SOA principles, i.e. oriented by the offer and request of services, to fulfil industrial systems goals. An important note is that these service-oriented agents do not only share services as their main form of communication, but also complement their own goals with external provided services.

2.3. Web Services

The W3C defines the concept of Web service (WS) and offers a family of WS-* standards to support the implementation of concrete SOA applications using the Internet as communication path between service consumers and service providers. As stated in the Web Services Glossary [W3C, 2004]: "*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). 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.*" This inflexible view, requiring a specific implementation, elides the existence of other alternative Web services technologies like the Representational State Transfer (REST) [Fielding, 2000] approach (to be seen at the end of this subsection).

As stated, W3C Web services use XML messages following the SOAP standard [W3C, 2007a] that defines the message structure and is the base of service interoperability (dealing with requests and responses). The common grammar used to describe the Web services is determined by the Web Services Description Language (WSDL) [W3C, 2007b]. A client connecting to a Web service reads the WSDL file to determine the available operations.

Web services awareness typically is achieved by using a registry. UDDI [OASIS, 2004] provides the infrastructure required to publish and discover services. The concomitant use of SOAP, WSDL and UDDI allows to implement the more usual interaction schema for Web services, as shown in Figure 2.

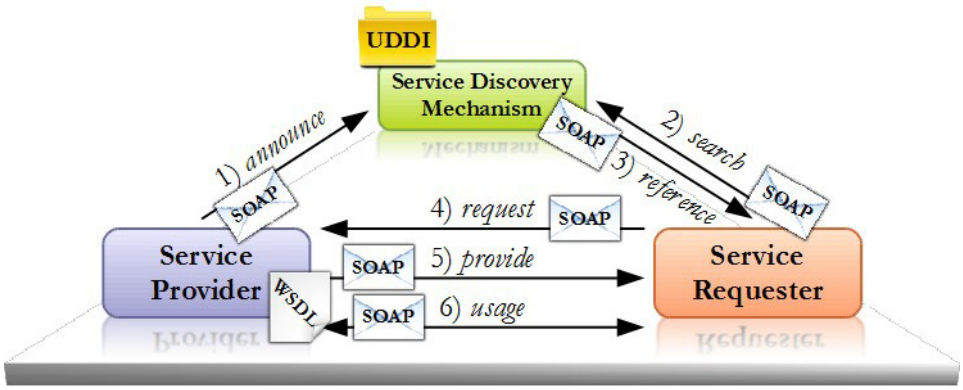

Figure 2: SOA triangle with core Web services protocols [Mendes, 2011].

Initially, a service provider requests the registration of a service into a UDDI registry service using a SOAP message (1). A service requester searches the service (2) and the UDDI registry sends a reference of the service (3). The requester calls the service using the reference (4). Then, after acceptance of the request (5), the interaction to consume the service starts (6).

With the emergence of numerous embedded devices with processing capabilities and connection to the Internet has become natural to think on their participation in SOA solutions. To cope with that reality, the Devices Profile for Web Services (DPWS) framework defines a minimal set of implementation constraints to enable secure Web service messaging, discovery, description, and eventing on resource-constrained endpoints [OASIS, 2009].

Languages for the description of complex operations allowing the composition of atomic Web services are available. When business processes are being exposed as Web services, the creation of new composite services using orchestration mechanisms may be described using the Business Process Execution Language (WS-BPEL) [OASIS, 2007]. On the other hand, choreography compositions (peer-to-peer collaborations) can be described using a choreography language like the Web Services Choreography Description Language (WS-CDL) [W3C, 2005].

The Semantic Web is an extension of current Web in which information is given with a well-defined meaning, allowing the performance of cooperative work between computers and people [Agarwal, 2012]. A step towards the Semantic Web is the use of Semantic Web services [Rodrigues, 2013]. Despite WSDL does not include semantics in the description of Web

services, the Semantic Annotations for WSDL and XML Schema (SAWSDL)[10] [W3C, 2007c] allows to add annotations to WSDL elements with additional semantic content.

More WS/SOAP-related standards are available from W3C and OASIS (see Table 2 with examples) but will not be addressed in this work.

Standard	Description	Organization
Web Services Addressing 1.0 - Core	Provides transport-neutral mechanisms to address Web services and messages.	W3C
Web Services Context (WS-Context) v1.0	Provides a definition, a structuring mechanism, and service definitions for organizing and sharing context across multiple execution endpoints.	OASIS
Web Services Dynamic Discovery (WS-Discovery)	Defines a discovery protocol to locate services. In an ad hoc mode of operation, probes are sent to a multicast group, and target services that match return a response directly to the requester.	OASIS
Web Services Federation Language (WS-Federation)	Defines mechanisms to allow different security realms to federate, such that authorized access to resources managed in one realm can be provided to security principals whose identities and attributes are managed in other realms.	OASIS
Web Services Policy 1.5 - Framework	Provides a general purpose model and corresponding syntax to describe the policies of entities in a Web services-based system.	W3C
WS-Reliability v1.1	SOAP-based protocol for exchanging SOAP messages with guaranteed delivery, no duplicates, and guaranteed message ordering.	OASIS
Web Services Security v1.1.1	Set of documents addressing security issues related with Web services.	OASIS

[10] SAWSDL was based on W3C member submission Web Service Semantics (WSDL-S). Other submissions related with Semantic Web services were not considered by the W3C, namely OWL-S (http://www.w3.org/Submission/OWL-S/), WSMO (http://www.w3.org/Submission/2005/06/), and SWSF (http://www.w3.org/Submission/SWSF/). WSMO-Lite (http://www.w3.org/Submission/2010/05/) is currently under consideration.

WS-Trust v1.4	Provides a framework for requesting and issuing security tokens, and to broker trust relationships.	OASIS

Table 2: Examples of W3C and OASIS standards.

REST principles can be used to implement an alternative to SOAP-based Web services, the so called RESTful Web services, currently used by big enterprises like Yahoo, Google, and Facebook [Rodriguez, 2008]. REST is an architectural style, initially proposed by [Fielding, 2000], where data and functionality are identified using Uniform Resource Identifiers (URIs). All REST requests are stateless, meaning that they are independent of previous ones (no memory) and contains all the necessary information to make themselves understood at the destination. Data and functionality are collectively called resources. Resources are manipulated using a uniform interface considering the create, read/retrieve, update and delete operations, mimicking HTTP PUT, GET, POST and DELETE methods [Pautasso et al., 2008]. Despite requests always be stateless, server responses can be statefull, meaning that the response to a request is labelled as cacheable or non-cacheable [Fielding, 2000].

2.4. Enterprise Service Bus

SOA systems can be realized by an ESB that provides a layer on top of an implementation of an enterprise messaging system [Ziyaeva et al., 2008], as shown in Figure 3.

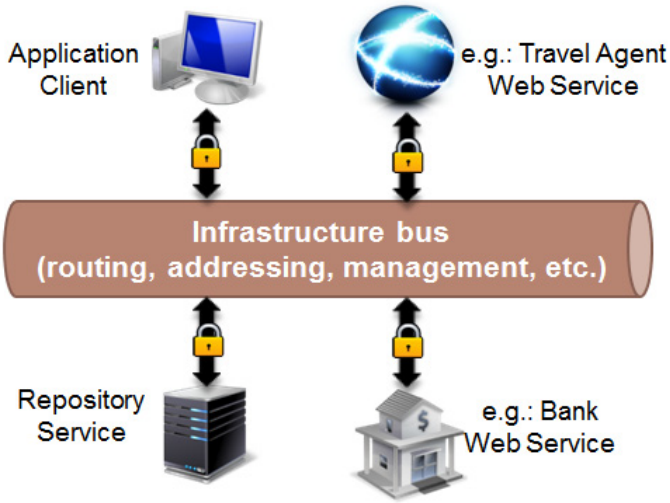

Figure 3: Enterprise Service Bus architecture.

Several ESB products are available, in the form of commercial and open source products (see Table 3), offering a variety of choices.

Product	**Url**	**Licensing**
JBoss ESB	http://jbossesb.jboss.org/	LGPLv2.1
Microsoft Azure Service Bus	https://azure.microsoft.com/en-us/services/service-bus/	Commercial
Mule ESB	http://www.mulesoft.org/	CPAL-1.0
Oracle Service Bus	http://www.oracle.com/technetwork/middleware/service-bus/overview/index.html	Commercial
Petals ESB	http://petals.ow2.org/	LGPLv2.1
Red Hat JBoss Enterprise Application	http://www.redhat.com/en/technologies/jboss-middleware/application-platform	Commercial
TIBCO ActiveMatrix BusinessWorks	http://www.tibco.com/products/automation/application-integration/activematrix-businessworks/default.jsp	Commercial
WebSphere Enterprise Service Bus	http://www-03.ibm.com/software/products/en/wsesb	Commercial

Table 3: Some existing ESB products.

Interoperability between heterogeneous systems requires information exchange. To be able to communicate, each participant must understand the contents of incoming messages. Writing “glue code” to deal with the translation of data between systems is a case by case approach, since the inclusion of a novel system originates writing more “glue code”. This approach is time consuming and constitutes a major problem when dealing with solutions that needs to scale. To deal with the translation of messages, ESB solutions provide specialized adapters (Figure 4) offering the possibility of linking heterogeneous systems.

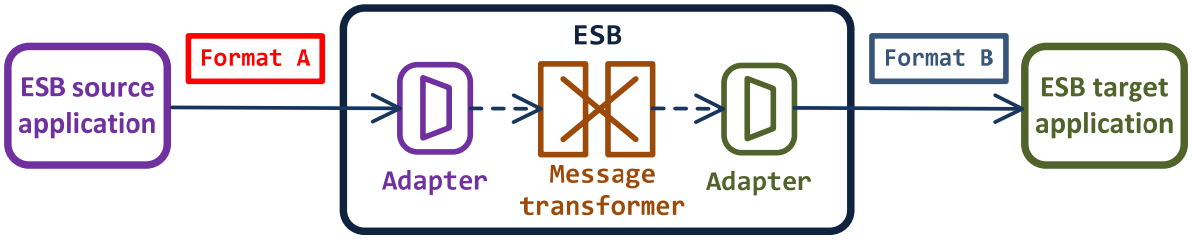

Figure 4: ESB message translation.

In addition to protocol translation, other typically desirable capabilities of ESBs include, without being exhaustive, Web service integration, process orchestration (typically via WS-BPEL), hot deployment, versioning, lifecycle management and security (e.g. message encryption) [DiMaggio et al., 2012]. Conceptually, when using an ESB, everything is either a service (not necessarily a Web service) or a message being sent between services. Synchronous and asynchronous communications between services is available, the latter being the most convenient to implement SOA compliant solutions.

It is recognized the importance of SOA type solutions to provide monitoring and management capabilities at service level [Wang et al., 2012]. Currently the ESB platforms only provide basic functions associated to the service registry (see Table 4 with an example for a concrete solution) and completely misses more advanced functionalities.

Feature	**Description**	**JBoss ESB support**
Inquiry capabilities	Allows to perform searches in order to find a service.	Yes
Publication capabilities	Allows the publication of services.	Yes
Replication capabilities	Permits the copy of information between UDDI nodes.	Yes
Registry federation	Allows to interconnect different ESB registries.	Yes
Security policy	Permits the use of authentication tokens for accessing services.	Yes
Subscription	Enables subscribers to “subscribe” to a UDDI registry, so they will receive information	Yes

	on changes made to the registry.	
Life-cycle monitoring and management of deployed services	If services fail or move elsewhere, their EPRs that may reside within the registry will remain until they are explicitly updated or removed by an administrator.	No

Table 4: Current and desirable features of JBoss ESB registry.

At service level, an OASIS candidate standard specifies the following sub quality factors: response time, maximum throughput, availability, accessibility and successability (see [OASIS, 2012] for more details). Service life cycle is also discussed in a W3C note [W3C, 2004c]. Despite being good starting points to establish some KPIs (Key Performance Indicators) and to describe some of the possible states for a service, complementing the monitoring and management capabilities with more advanced features allowing to detect in advance the possible degradation or risk propagation is a step forward.

Chapter 3

Life-cycle Management Module Specification and Implementation

ARUM is a collaborative project within the European Commission "Factory of the Future" initiative and is funded under the 7th Framework Programme. The project addresses the development of novel Information Communication Technology (ICT) solutions to handle new challenges in the ramp-up production of complex and highly customized products, such as aircraft and shipbuilding industries. This chapter addresses the specification and implementation of the Life-cycle Management Module, enlightening its contribution to fulfil the ARUM project objectives.

3.1. Positioning in the ARUM Project

The ARUM project focus is on the development of mitigation strategies to respond faster to unexpected events and the implementation of systems and tools for the decision support in planning and operation. For this purpose, the ARUM platform comprises an intelligent ESB (iESB), which enriches the traditional ESBs with a plethora of advanced modules, and provides a common infrastructure for the integration of heterogeneous planning and scheduling tools (e.g., using the MAS principles) and legacy systems, as illustrated in Figure 5. The main modules deployed in the iESB are the Ontology service, Data Transformation Service, Sniffer, Node Management, and Life-Cycle Management (note that only the latter is the responsibility of the author of this work).

The Ontology service module is responsible to gather the pieces of data from various legacy systems, e.g., MES (Manufacturing Execution System) and ERP (Enterprise Resource Planning), via the data transformation service, aggregate and store it in the local triple store

and then provide it on request to other services. Aiming to provide a common and explicitly defined semantics of data, it was developed a set of OWL (Web Ontology Language)-based ontologies for the description of production processes, shop floor topologies, resources and their availability, scheduling strategies and disruption events [Inden et al., 2013].

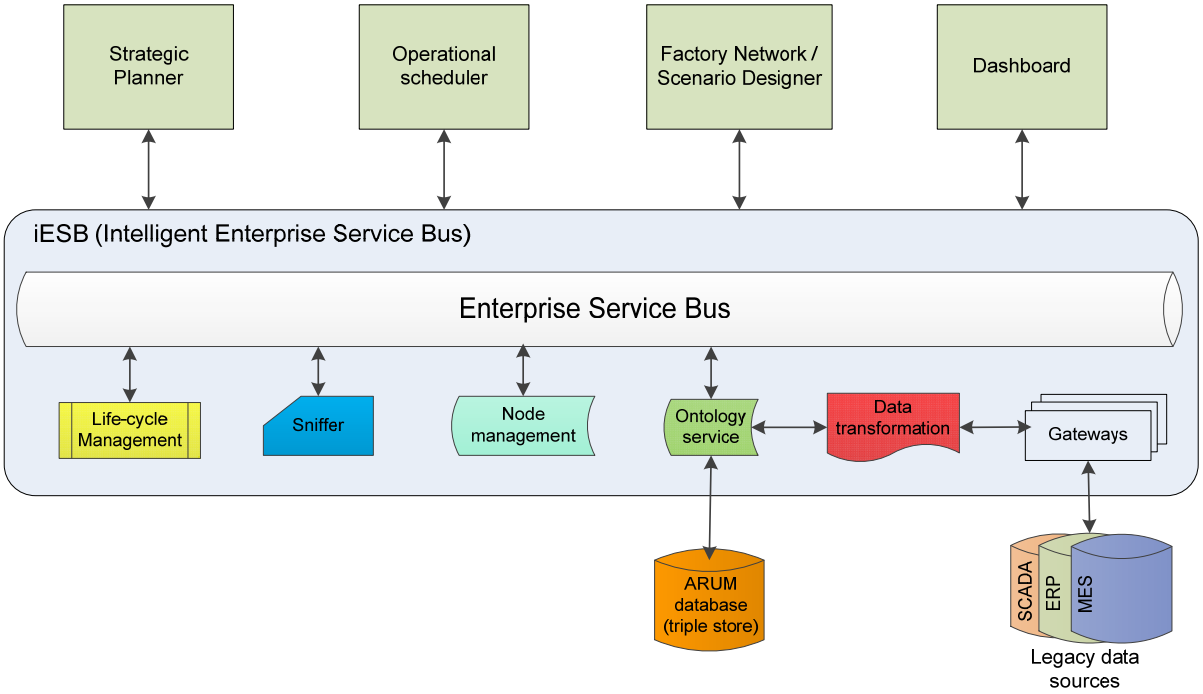

Figure 5: ARUM platform high level architecture (adapted from [Leitão et al., 2013]).

The Data transformation service module is responsible for gathering data from legacy systems. The raw data, received from gateways using the legacy system specific interfaces and communication protocols, is transformed into the ontological format (RDF – Resource Description Framework) using the OWL-based ontologies provided in the Ontology service.

The Sniffer module is responsible for capturing the flow of messages across the ESB and related to the registered services, to support the monitoring and understanding of the overall state of the system especially in a distributed environment with multiple interacting services [Vrba et al., 2014].

The Node Management module supports the distributed management of iESB instances, allowing the inter-connection among several ESBs [Marín et al., 2013].

The Dashboard acts as a user interface (UI), providing the user with the means for administration and monitoring of the overall ARUM solution (including all deployed tools). It means for example the deployment of services, monitoring their parameters and health, visu-

alizing the message flow and statistics. The dashboard leverages the web portal technology, which is a specially designed web page on which the information is displayed within dedicated user interface components – the portlets.

The Life-cycle management module performs the life-cycle monitoring and analysis of the health of the services that are deployed within the iESB, supporting the dynamic, online and on-the-fly actions to mitigate the degradation of their performance. This module will be deeply analysed during the rest of this document.

3.2. LCMM Architecture and Functions

The Life-cycle Management Module (LCMM) performs the continuous monitoring and data analytics of the services that are deployed within the iESB, allowing to dynamically be aware of the current state and health of the services and to perform on-the-fly actions to increase the services' performance. In particular, the main features provided by the LCMM module are:

- Monitoring of the registered services' health, providing on-line information related to different KPIs, such as the failure rate and the occupancy.
- Detection of the registered services/tools that are not operating properly and analysis of trend and patterns on the services' performance, e.g., the detection of the degradation in the service quality.
- Analysis of the risk propagation in case of service quality degradation.
- Suggestion of actions to maintain the system's robustness and stability.

The LCMM module interacts with the Sniffer module to get data related to the exchanged messages and the UI Dashboard to support the interaction with the user and particularly to display the monitored info related to the health of registered services according to pre-defined KPIs, as illustrated in Figure 6. Internally, the module comprises the Event Monitoring, Data Analysis and local database.

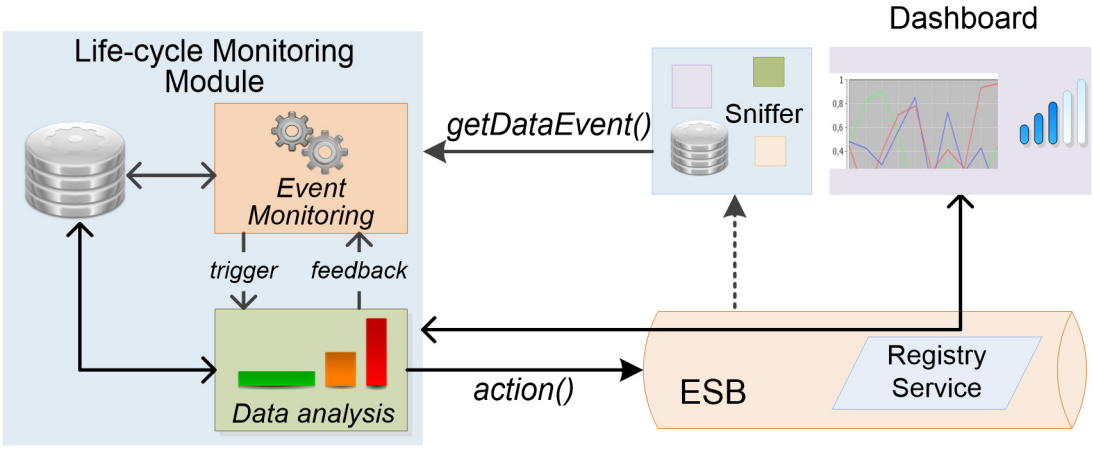

Figure 6: Architecture of the life-cycle management module.

The interaction between the Event Monitoring and Data Analysis components allows to trigger a more detailed data analytics and also to provide feedback regarding the adjustment of the pooling rate for a specific service.

3.2.1. Event Monitoring Component

The Event Monitoring component performs mainly the collection of the data related to the exchanged messages across the bus and the monitoring of the services' health. Since the Sniffer module is continuously sniffing the messages crossing the ESB and feeding its database with the gathered information, the Event Monitoring component can request this data using a proper and dynamic polling mechanism that is parameterized according to the service frequency and priority. In fact, the polling time is adjusted according to the service usage frequency, i.e. short polling time if the service is usually requested or larger time if rarely requested. Also, an event-driven mechanism can be used to collect the data from the Sniffer module, but this alternative can only be used if the Sniffer module provides the subscription functionally.

The reasoning engine, embedded in this component, processes the gathered and historical information in order to support the health monitoring of registered services by calculating several pre-defined KPIs, namely in terms of performance and status, that will be exposed as monitoring services to the user, namely through the UI dashboard. Examples are the detection if the registered services are not alive by identifying not answered messages and behaviours that not follows the service patterns.

Considering that $T = \{tool_i : i = 1, \dots, M\}$ is the set of tools connected to the ESB and each tool offers a set of services $S_i = \{service_{ij} : j = 1, \dots N_i\}$, the LCMM module provides a plethora of services aiming to monitor several KPIs, as detailed as follows (also illustrated in Figure 7):

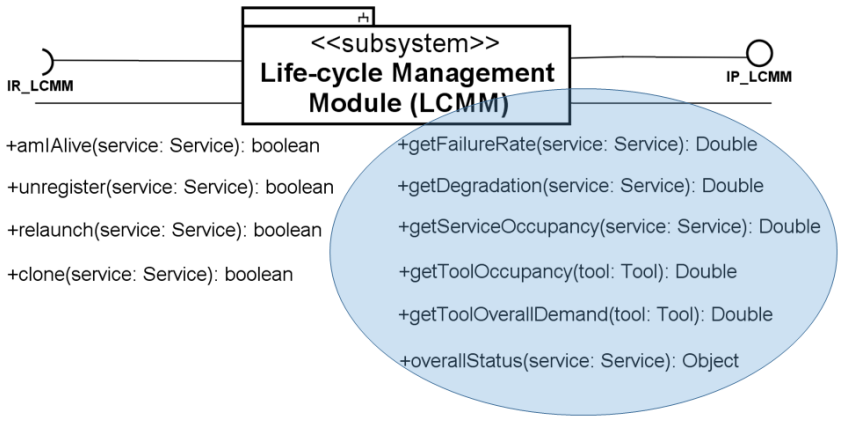

Figure 7: Services provided by LCMM.

- *getFailureRate*: provides the failure rate of a service, calculated as follows, where f_{ij} is the number of failures of the $service_{ij}$ with reference to the last n requests of this service:

$$FR_{ij} = \frac{f_{ij}}{n} \quad (1)$$

- *getDegradation*: provides the information related to the degradation of the response time of a service j of the tool i (δ_{ij}). The degradation is the comparison of the response time of the last two events.

$$D_{ij} = \delta_{ijt} - \delta_{ij(t-1)} \quad (2)$$

- *getServiceOccupancy*: provides the information related to the occupancy of a service. The Service Occupancy (SO_{ij}) of a service j running in the tool i is defined as the ratio of the overall time t_{ij} that the service is being used with the overall time Δ_i of the software tool deployed on the system:

$$SO_{ij} = \frac{t_{ij}}{\Delta_i} \quad (3)$$

- *getToolOccupancy*: provides the information related to the occupancy of a tool. The Tool Occupancy (TO_i) is defined has the ratio of the time t_i that a given tool i is being

used (independent of the overlapping of services in the tool) with the overall time Δ_i of the tool deployment on the system, as illustrated as follows:

$$TO_i = \frac{t_i}{\Delta_i} \quad (4)$$

- *getToolOverallDemand*: provides the information related to the load of a tool within the overall ESB load. This load is the ratio of the number r_i of requests to the services running in tool *i* and the total number of requests to all tools, represented as follows:

$$TOD_i = \frac{r_i}{\sum_{k=1}^{M} r_k} \quad (5)$$

- *overallStatus*: provides the overall service status considering all evaluation parameters, namely the failure, degradation and occupancy, weighted according to pre-defined values. This can be defined as a health scale, where 0 means "good", 1 means a potential "risk" or "problem".

This component can also implement a pre-risk analysis allowing to determine potential situations of service/tool failure. In this way, when a set of conditions are met, such as the presence of historical problematic tools or the warnings coming from the evolution of service KPIs, the component can signalize the critical service(s) and take more pro-active measures, such as changing the warning threshold values. Beside this action enabling the early signalling of potential hazardous situations, it additionally allows an anticipated taking of known actions that permit to overcome the potential situation.

3.2.2. Data Analysis Component

The Data Analysis component aims to perform advanced reasoning, and particularly data analytics, over the historical and current collected information related to the deployed services. In fact, the functions provided by this component include:

- Analysis of trends to detect deviations or patterns in the quality and performance of the service.
- Analysis of correlation among the execution of different services (also including the correlation considering services deployed in other ESBs belonging to the same network).
- Analysis of the impact and risk propagation related to the degradation of a service.

The implementation of these functionalities may consider the use of data mining techniques [Witten et al., 2011], namely clustering algorithms. Clustering is a technic used to find, in an automatized way, hidden patterns in big quantities of data. Based on the *k*-means clustering algorithm presented in [Kanungo et al., 2002], Figure 8 illustrates a strategy integrated in LCMM to perform data analytics to discover the set of services that presents more risk of abnormal behaviour.

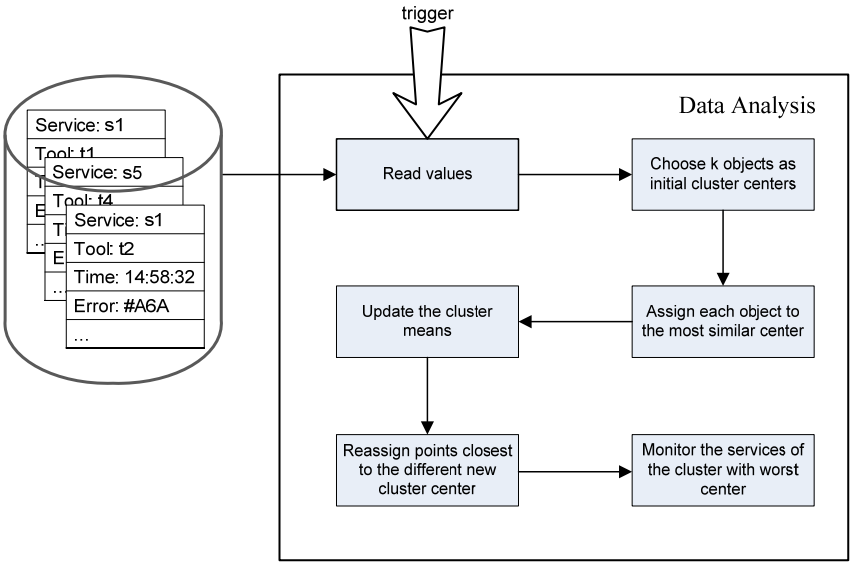

Figure 8: Algorithm for detecting patterns in the data analysis.

A periodic or trigger event causes the Data Analysis component to start reading the values collected by the Event Monitoring component in local database. The algorithm prescribes choosing *k* arbitrary readings as the cluster centres. All readings are assigned to the most similar centre based on the calculation of Euclidean distance between the read values. A next step is related to the calculation of the most central point for each cluster (not necessarily equal to the initial choice for the centre). Then, it is performed a new reassigning of all values in conformity with the new centres. Finally, the services in the cluster with the worst centre are signalized as services with possible degraded performance, and in this way will require a close monitoring.

The output of these functions is the generation of warnings to the user, e.g., providing useful information about the state and risk of a specific service and also suggesting the execution of proper actions, such as unregistering the service (e.g., when the service is not being used), re-starting of a tool (e.g., when the service is degraded or not responding) or creating one clone (e.g., when the service/tool is too busy). These actions can also be

performed automatically, under well controlled conditions. In this case, as also illustrated in Figure 9, and aiming to support the operation of the LCMM module, several services provided by other modules in the iESB may be requested, namely *amIAlive* (to verify if the service is alive), *unregister* (to unregister services by accessing the ESB Register Service), *relaunch* (to restart the service provided by the tool) and *clone* (to clone a service/tool, e.g., when a service/tool is too busy). Note that the use of these services may require some kind of privilege access to external tools.

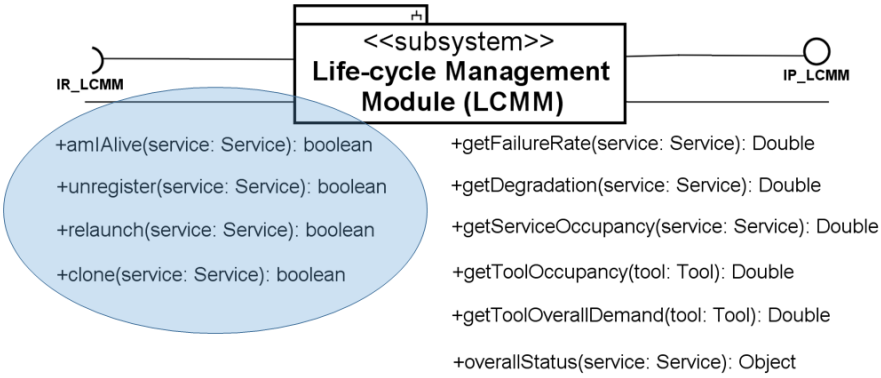

Figure 9: LCMM requested services.

Learning is an important piece of the LCMM module, supporting the discovery in advance of potential problems and the definition of the actions to be implemented when a risk is detected (as well as in the adaptation of the warning threshold values). The LCMM module should also consider self-monitoring and self-analysis in order to avoid its chaotic behaviour, e.g., acting as a "cancer" deploying very rapidly services/tools and consequently overloading the system.

3.3. LCMM Contribute to Robustness and Self-organization

The implementation of the LCMM functionalities is a step forward to achieve intelligence in the ESB platform and in this way to achieve an iESB. More concretely, this module may contribute to achieve robustness, reliability, fault-tolerance and self-organization in this kind of distributed systems, i.e. those based on the ESB middleware.

Robustness can be defined as the capability of a control system to remain working correctly

and relatively stable, even in presence of disturbances [Pereira et al., 2013]. Additionally, an important issue is the system fault-tolerance, i.e. the capability to detect and tolerate internal failures, in order to continue performing their operations without the need for an immediate intervention. Being more tolerant, the downtime is reduced, and being able to detect and diagnosis, the repair process is speed-up, increasing the robustness and productivity of manufacturing systems [Leitão, 2011].

In such kind of distributed systems, based on offering and requesting services, the inexistence of central nodes makes these systems more robust than the traditional centralized systems, by eliminating the single point of failure problem. In fact, more decentralization provides additional reliability due to the implicit redundancy and diversity and the non-dependency of central control nodes [Pereira et al., 2013]. However, the existence of a middleware infra-structure to integrate the IT software applications based on services can somehow restrict the robustness and reliability of such systems. Note that reliability is the ability of a system or component to perform its required functions under stated conditions for a specified period of time. In this way, the LCMM module ensures the increase of robustness and reliability by permitting an automatic discovery of problematic services, e.g., the ones that may be failing, not responding properly or overloaded, and take/suggest appropriate actions, such as launching a parallel service of the one that is identified to be near of failure, to mitigate the possible problems.

Additionally, the LCCM module can greatly contribute as an underlying mechanism to support self-organization at two levels: at service level or at iESB level. On the first case, the LCCM module can act as a referee, issuing warning signals for the deployed services, preventing erratic behaviour (e.g., when a tool is sending over the limit service requests). In this case, and if the appropriate behaviour actions are implemented in the affected tool, the tool can change its internal behaviour accordingly. A second example can be found in a tool that has reduced utilization. In this case, and if a redundant tool is presented, the LCMM can advise the less used tool to change into low profile mode or to, at the limit, un-plug itself from the system, as seen in Figure 10 (hexagonal service).

At iESB level, the information mined by the LCMM can be used by the self-organization mechanism as a way to internally (re)arrange structurally the iESB, by adding, modifying or removing services (ellipse service in Figure 10), or (re)arranging the relations and constitution of clusters in an inter-iESB perspective. This structural self-organization level allows the

dynamic clustering of iESBs, arranging themselves accordingly, aiming an uniform service performance distribution where the performance of each individual iESB is increased by the decrease of individual service/tool overload and failure rate.

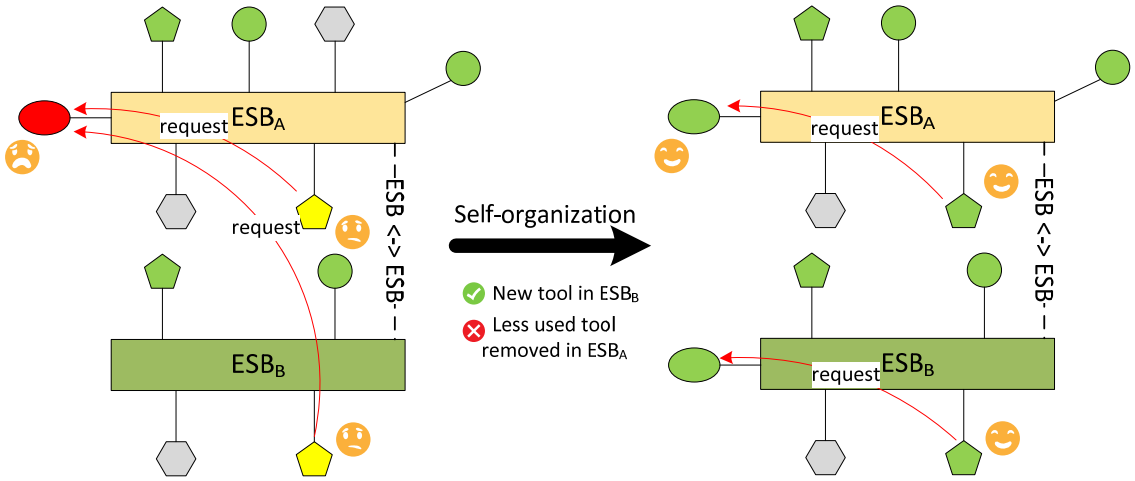

Figure 10: Structural self-organization in a network of ESB clusters.

The aforementioned insights are drawn from the ADACOR^2 control architecture. In the proposed architecture, the individual behaviour of the entities [Barbosa et al., 2013a] are dynamically changed, aiming a smooth evolution of the system, while a more drastic evolution is achieved through the change of the entities relations [Barbosa et al., 2013b]. Similarly to what is achieved in the ADACOR^2 approach, by combining these two self-organization levels, the LCMM module will enable the achievement of a self-organized and evolvable iESB system, once the overall system is able to adapt itself internally and structurally to system demand fluctuations, internal services disruption or to iESB node change.

Additionally, and also as indicated in the ADACOR^2 control architecture, the LCMM must undergo with a nervousness controller in order to avoid entering in a chaotic process when taking decisions. This stabilization mechanism will prevent intermittent service/tool stop or launch as also the constant (re)arrangement of the iESB clustering.

3.4. LCMM Implementation and Operation

The proposed LCMM was developed and deployed as a JBoss ESB service, encapsulating its business logic into a set of Java classes. JBoss ESB [DiMaggio et al., 2012] is an ESB solu-

tion maintained under the umbrella of JBoss Community[11] and intends to provide an open source option for the construction of systems based on SOA principles.

The main constitutive part of the LCMM service is a chain of "Actions". Basically, in the JBoss ESB framework, an "Action" is a Java class that allows the ESB services to carry out their tasks:

```
package pt.ipb.arum.lcmm;
...
public class LcmmAction extends AbstractActionLifecycle {
    ...
    public Message process(Message message) {
        Message esbMessage =
              MessageFactory.getInstance().getMessage();
        esbMessage.getBody().add(engine.getSnapshots());
        return esbMessage;
    }
}
```

These tasks are realized after the processing of the data referring to the exchange of messages between the registered services in the ESB. To accomplish that, it was implemented a connection to the Sniffer's database, which is implemented using a MySQL database. Then a snapshot is created for each service and tool:

```
public Snapshot(String toolName, String serviceName) {
    this.timeStamp = new Timestamp(System.currentTimeMillis());
    this.toolName = toolName;
    this.serviceName = serviceName;
    this.failureRate = failureRate();
    this.degradation = degradation();
    this.serviceOccupancy = serviceOccupancy();
    this.toolOccupancy = toolOccupancy();
    this.serviceOverallDemand = serviceOverallDemand();
    this.toolOverallDemand = toolOverallDemand();
}
```

[11] https://www.jboss.org/

For example, below are shown some details of the implementation of `failureRate` method used to create a snapshot.

```
private double failureRate() {
    ResultSet resultSet;
    int requests = 0;
    int replies = 0;
    double res = 0.0;
    try {
        resultSet =
            SnifferDB.getInstance().query("select count(*) ...");
            ...
                    requests = resultSet.getInt(1);
    ...
    try {
        resultSet =
            SnifferDB.getInstance().query("select count(*) ...");
            ...
                    replies = resultSet.getInt(1);
    ...
    if(requests != 0) {
        res = ((requests - replies) * 1.0) / (requests * 1.0);
    }
    return res;
}
```

The Weka[12] solution, a data mining and machine learning API, was used to implement the Data Analysis component.

```
...
    InstanceQuery instanceQuery = new InstanceQuery();
    instanceQuery.setDatabaseURL("jdbc:sqlite:./lcmm.db");
    instanceQuery.setQuery("select ...");
    Instances instances = instanceQuery.retrieveInstances();
...
    SimpleKMeans simpleKmeans = new SimpleKMeans();
...
```

[12] http://www.cs.waikato.ac.nz/ml/weka/

The LCMM integrates with the ESB by using a configuration file, `jboss-esb.xml` that appears below in a partial view:

```
<?xml version="1.0"?>
<jbossesb parameterReloadSecs="5"
...
  <providers>
    <jms-provider connection-factory="ConnectionFactory"
      name="JMS">
      <jms-bus busid="serviceJMSChannel">
        <jms-message-filter dest-name="queue/lcmm_ESB_request"
          dest-type="QUEUE" />
...
  <services>
    <service category="ARUM-Admin" description="ARUM Life-Cycle Manage-
ment Module"
      name="LCMM">
      <listeners>
        <jms-listener busidref="serviceJMSChannel" name="ESBListener" />
      </listeners>
      <actions>
        <action class="pt.ipb.arum.lcmm.LcmmAction" name="LcmmAction"
          process="process" />
        <action name="notificationAction"
class="org.jboss.soa.esb.actions.Notifier">
          <property name="okMethod" value="notifyOK" />
          <property name="notification-details">
            <NotificationList type="ok">
              <target class="NotifyTopics">
                <topic jndiName="topic/lcmm_JMS_topic" />
...
```

The user interface, supporting the visualization of the data resulting from the processing operation of the LCMM service, was developed as a web-based application that can be accessed via web browser. This web-based application was built on the Liferay Portal[13] [Sarang, 2009] as a portlet. One of the central pieces used to construct the portlet was the Highcharts

[13] http://www.liferay.com/

3.0[14], which is a charting library written in HTML5 and JavaScript, allowing, among others, to build dynamic charts. Next javascript code shows the instantiation of the overall status chart:

```
<script type="text/javascript">
(function($){ //overallStatus
    $(function(){
        ...
        $('#overallStatusChart').highcharts({
        xAxis:{categories:[
            'Failure Rate',
            'Degradation',
            'Occupancy',
            'Overall Demand'],tickmarkPlacement:'on',lineWidth:0},
        ...
})(jQuery);
...
```

The communication between the LCMM service and the Liferay portlet is achieved by using the Java Message Service (JMS). The JMS specification describes the exchange of messages between Java programs, in particular for the use in publish-subscribe solutions. Going into details, it is used a JMS topic, allowing the delivery of messages to multiple subscribers:

```
<?xml version="1.0"?>
<configuration xmlns="urn:hornetq"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:hornetq /schema/hornetq-jms.xsd">
    <queue name="lcmm_ESB_request">
        <entry name="queue/lcmm_ESB_request"/>
    </queue>
    <topic name="lcmmtopic">
        <entry name="topic/lcmm_JMS_topic"/>
    </topic>
</configuration>
```

[14] http://www.highcharts.com/

Figure 11 illustrates a screenshot showing an overview of four KPIs related to the evolution of two services, namely the failure rate, the degradation, the occupancy and the overall demand.

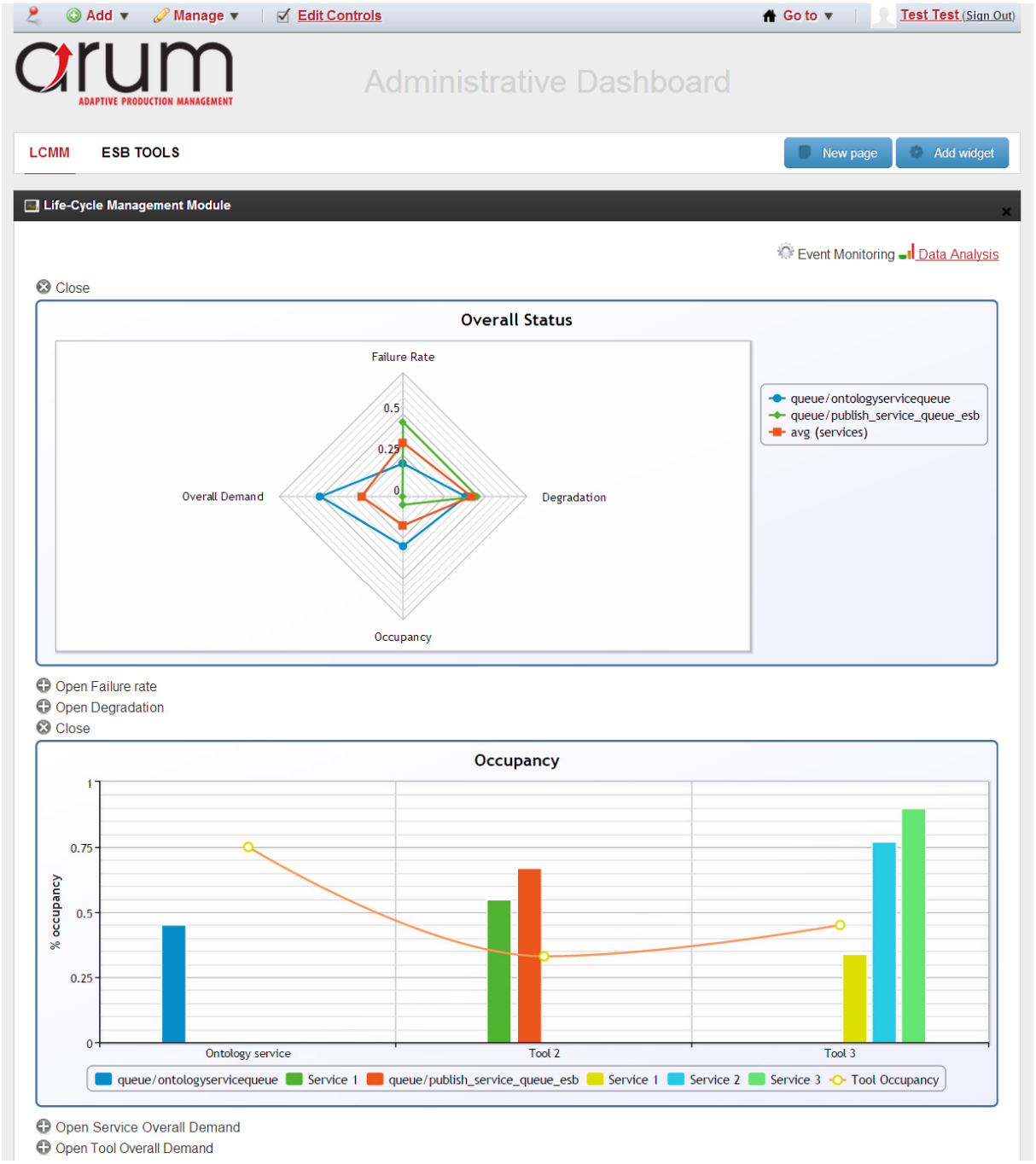

Figure 11: Screenshot of the LCMM user interface.

For each KPI, a chart with their evolution is shown. In the case of occupancy index, it is presented a joint view of services and tools. The continuous monitoring of these KPIs allows to

detect problems and to trigger warnings for the implementation of proper actions that will mitigate their possible negative impact.

The part of the GUI related to the Data Analysis presents a table, as can be seen in Figure 12. The entries of each KPI for each service are presented in the form `value [trend]`. The trend reflects the evolution of the indexes comparing with the previous instant and it is a value in the set `{equal, up, down}`.

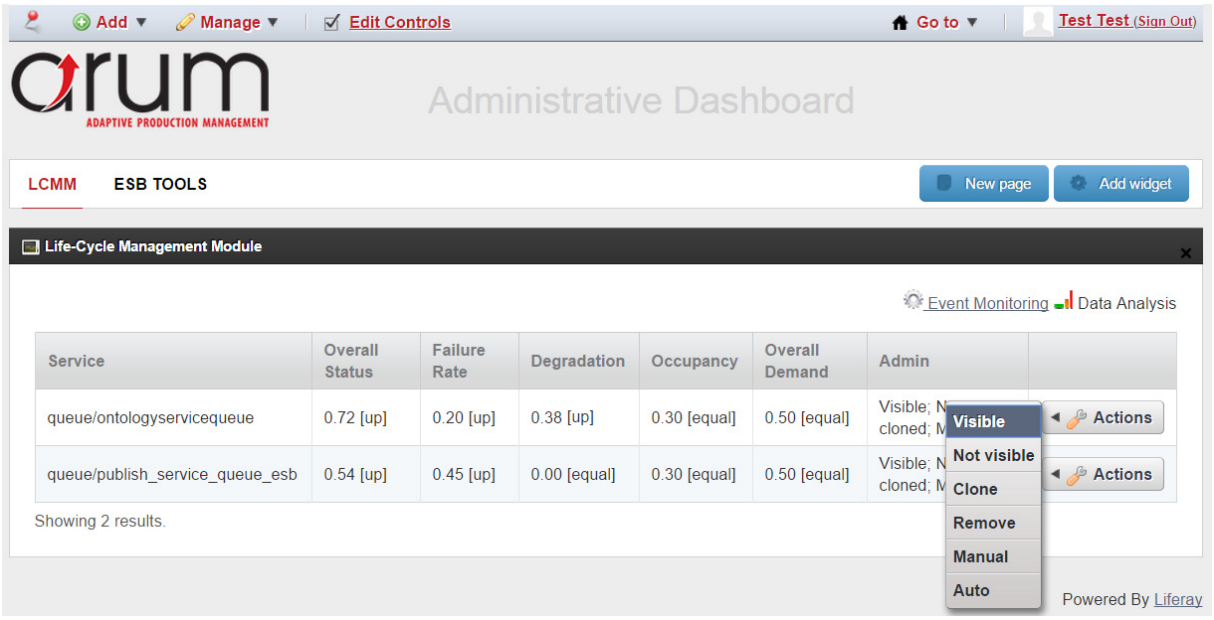

Figure 12: Detail of the LCMM user interface related to Data Analysis.

The “Actions” button, presented in the last column, allows to perform some administrative tasks linked to the service to which it relates. The options “Visible” / “Not visible” allow the user to decide whether to visualize the KPIs curves of the service in the Event Monitoring part. The “Remove” option permits to instruct the Node Management module to unregister / undeploy the service from the iESB. The "Clone" option instructs the Node Management module to trigger the installation of a new instance of the service in another iESB with the consequent removal of the service in the current iESB. These two features enable the system to self-organize when the “Automatic” mode is activated. The default option is the “Manual” mode. The column “Admin” allows to view the current state of the administrative options enabled for each of the services. More details and screenshots are presented in Chapter 4.

Chapter 4

Results and Discussion

This chapter presents two experimental use cases which demonstrate the advantages of the use of the LCMM towards the system's robustness and self-organization. The first one is a proof of concept that was presented and validated by the ARUM partners in the second year meeting review. In the second use case, it is possible to appreciate more advanced features of the module, namely the use of LCMM's Data Analysis Component to achieve structural self-organization in an ecosystem constituted by two iESBs.

4.1. Experimental Setup

The use cases were executed on a PC with the characteristics listed in Table 5.

Component	Description
Processor	Intel Core i5-3317U
Total memory	4 GB
Operating system	Windows 8 x64

Table 5: Hardware and operating system.

The used software components were the ones listed in Table 6.

Component	Version
Java	1.6 update 41

Ant	1.8.4
JBoss AS	6.1.0.Final
JBoss ESB	4.12
Liferay portal	6.1.1 GA2

Table 6: Versions of software components.

For the first use case one instance of the iESB was launched and for the second one two iESBs were launched.

4.2. Peak on Demand Use Case

In this use case, two services were deployed into the iESB: the “ontology service” and the “publish service”. The first was implemented to leave unanswered 1/5 of the requests and the second to fail 1/10 of the requests. Each of the services has been requested by a client every second. After 40 requests, the “publish service” began to fail 3/20 times and the requests doubles.

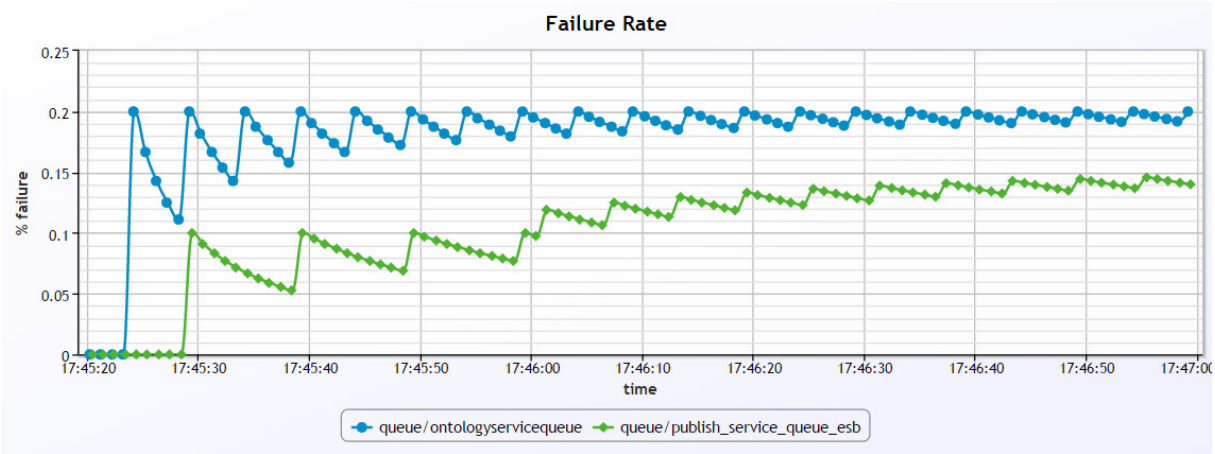

Figure 13: Failure rate evolution.

As showed in Figure 13, the failure rate of the “ontology service” converges to 20%, stabilizing around this value. Observing the evolution of the failure rate curve of “publish service”, it is possible to verify that, between 17:46:10 and 17:46:50, the failure rate is increased from 10% to 15%. In parallel (seeing the chart in Figure 14), it is possible to observe a degradation of the response time of the “ontology service” and “publish service” after 17:46:00, which may be explained by a peak on the demand using the bus.

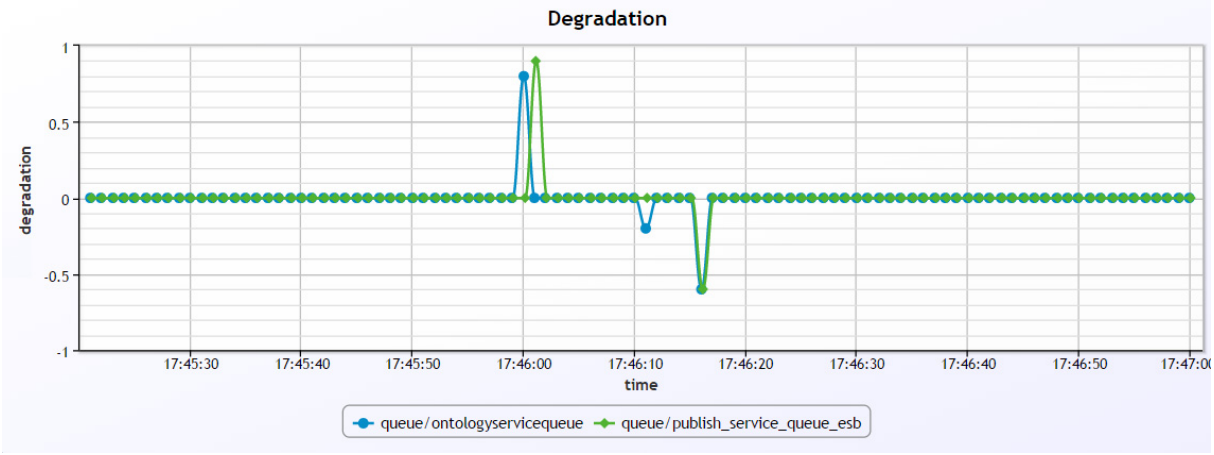

Figure 14: Degradation evolution.

However, both services recover after a while as shown by the negative values in the chart. The continuous monitoring of these KPIs allows to detect these problems and to trigger warnings for the implementation of proper actions that will mitigate their negative impact.

4.3. Inter-iESB Use Case

The objective of the second use case is to demonstrate the use of Data Analysis component. Two instances of the iESB were launched and joined using the Node Management module present in each of them. In the first iESB, 30 services were deployed, as illustrated in Figure 15.

Figure 15: JBoss ESB JMX Console with the deployed services in the first iESB.

The services s_i, $i = 10, 20, 30$ have been conditioned to fail 1/10 of the requests and for the other services were not forced any failure. For each service has sent a request every second. In the second iESB were running the natives services along with the LCMM and the Node Management module (see Figure 16).

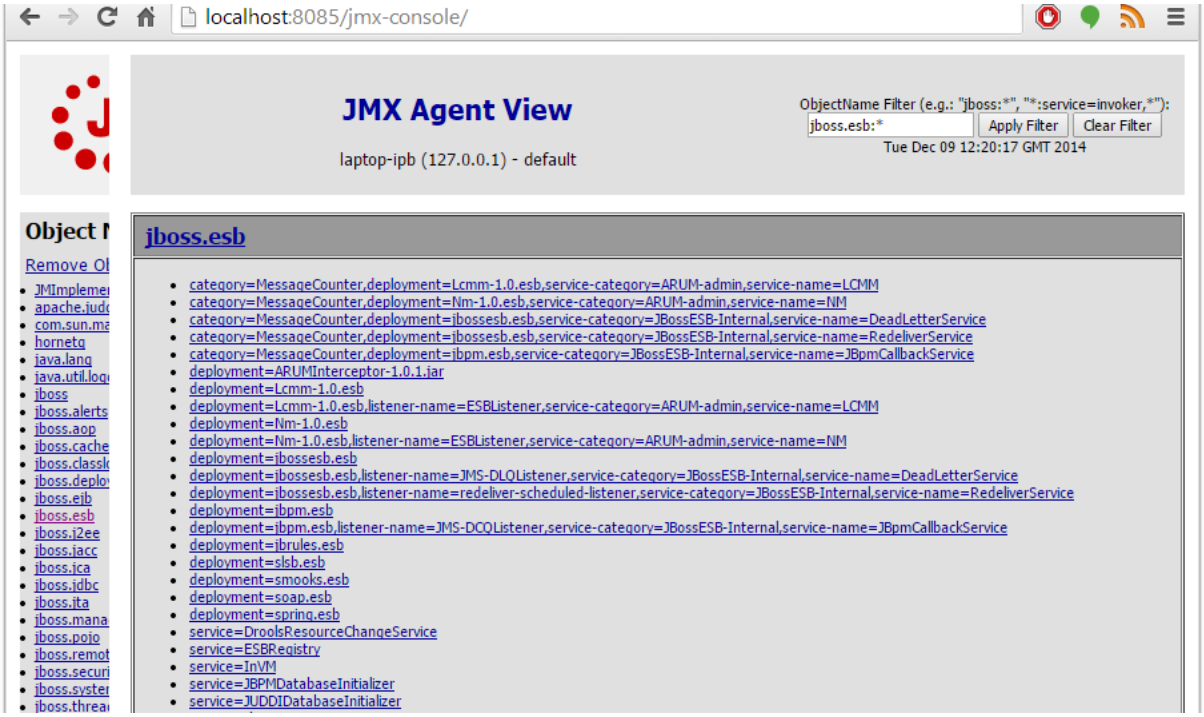

Figure 16: JBoss ESB JMX Console with the deployed services in the second iESB.

After letting the system to evolve for 10 minutes, it was made the screenshot of the LCMM module shown in Figure 17.

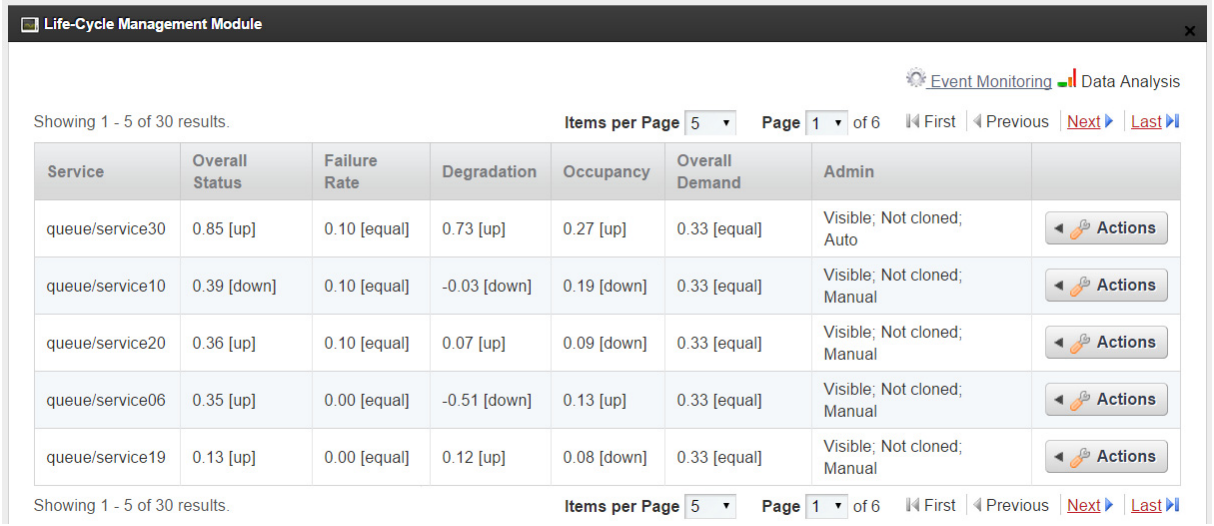

Figure 17: Screenshot of Data Analysis component.

It should be noted that the occupancy is increasing for the most problematic service (i.e. service 30) and that the automatic mode option was selected in order to be the system to address the recovery from problematic situations. In automatic mode, after detecting a problem, the LCMM sends messages to the Node Management module whenever appropriate. One of the possible requests is the uninstall of a service within a node (in this case in the first iESB) and the installation/registration of the service in another available node (in this case the second iESB). The Node Management module deploys and undeploy the packages containing the files of services by copy/delete involving the file system and the deployment folder of the nodes.

After setting the automatic mode for the most troublesome service, the system was allowed to evolve further 5 minutes. Figure 18 shows a detail of the list of services deployed on the second iESB. As can be seen, the service30 is now installed/registered in the second iESB.

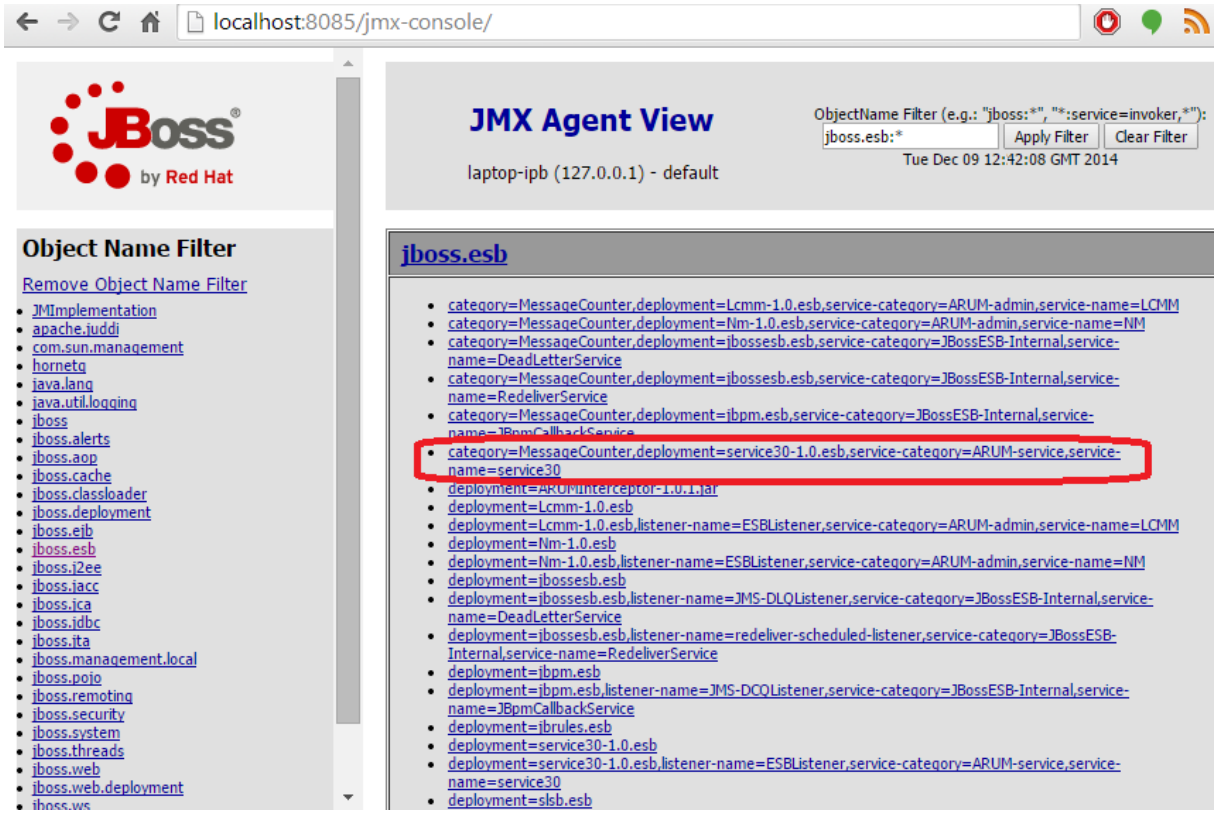

Figure 18: Services deployed in the second iESB after the system's self-organization.

This demonstrates the ability of the LCMM to detect problematic situations and trigger actions in Node Management module, allowing the system's self-organization in an automated way.

Chapter 5

Conclusions and Future Work

The use of ESB middleware allows to implement distributed systems that integrate loosely-coupled heterogeneous IT infra-structures. ESB provides several functionalities, namely the monitor and control of the routing of messages exchanged between software applications that expose their functionalities using services. Related to the life-cycle management of services, the ESB usually only provides basic functions associated to the service registry and completely misses advanced functionalities (e.g., data analytics).

5.1. Conclusions

This document described a service life-cycle management module that extends the traditional ESB features to provide advanced monitoring capabilities and data analytics to the registered services, contributing to achieve more robust and self-organized SOA-based systems.

The proposed module was implemented as a JBoss ESB service, using Java, and the user interface was developed as a web-based application built on the Liferay Portal. Several functions were implemented allowing to monitor the health of services according to pre-defined KPIs and also to detect trend and patterns in the service performance.

The experimental implementation allowed a proof of concept, by exploring two use cases related to the ARUM project. In the first one, it was observed the effect of a peak on demand and their detection by LCMM, enlightening the utility of the Event Monitoring component. In the second use case, it was observed the system's self-organization induced by LCMM after the detection of the services with less performance. Both use cases demonstrate the usefulness of LCMM to make the system more robust and self-organized.

5.2. Future Work

Approaching the end of the ARUM research project in late 2015, the partners' planning and scheduling tools will be finished and more data obtained in the real context will be available. Richer scenarios will be used in the ARUM Final Review Meeting that will be held at the Airbus industrial facility in Hamburg. This will allow the creation of other experimental use cases. On the other hand, the availability of more historical data will allow to perform massive tests using the developed module.

Another issue is related to the implementation of more powerful data mining techniques supporting the data analytics performed by the LCMM's Data Analysis Component. In parallel, the development of learning techniques will allow the system to make autonomous decisions in an automated manner.

The FIPA Quality of Service (QoS) Ontology [FIPA, 2002b] provides basic vocabulary for QoS related with the FIPA Message Transport Service [FIPA, 2002c]. Taking FIPA QoS as starting point, will be developed an ontology encompassing the KPIs presented in LCMM's Event Monitoring Component.

Bibliography

[Agarwal, 2012] P. R. Agarwal, "Semantic Web in Comparison to Web 2.0", in *Third International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, pp. 558-563, 2012.

[Barbosa et al., 2013a] J. Barbosa, P. Leitão, E. Adam, D. Trentesaux, "Self-Organized Holonic Multi-agent Manufacturing System: The Behavioural Perspective", in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC'13)*, pp.3829-3834, 2013.

[Barbosa et al., 2013b] J. Barbosa, P. Leitão, E. Adam, D. Trentesaux, "Structural Self-organized Holonic Multi-Agent Manufacturing Systems", in *Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS'13)*, Lecture Notes in Computer Science, vol. 8062, Springer pp. 59-70, 2013.

[DiMaggio et al., 2012] L. DiMaggio, K. Conner, M. B. Kumar and T. Cunningham, *JBoss ESB Beginner's Guide*. Packt Publishing, 2012.

[Erl, 2005] T. Erl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice-Hall, 2005.

[Fielding, 2000] Roy Thomas Fielding, Architectural Styles and the Design of Network-Based Software Architectures [Ph.D. Dissertation]. University of California, Irvine, 2000.

[FIPA, 2002a] Foundation for Intelligent Physical Agents (2002). *FIPA Abstract Architecture Specification* [Online]. Available at: http://www.fipa.org/specs/fipa00001/. [Accessed: November 2014].

[FIPA, 2002b] Foundation for Intelligent Physical Agents (2002). *FIPA Quality of Service Ontology Specification* [Online]. Available at: http://www.fipa.org/specs/fipa00094/. [Accessed: November 2014].

[FIPA, 2002c] Foundation for Intelligent Physical Agents (2002). *FIPA Agent Message Transport Service Specification* [Online]. Available at: http://www.fipa.org/specs/fipa00067/. [Accessed: November 2014].

[Gartner, 2002] Gartner, Inc. *Predicts 2003: Enterprise Service Buses Emerge*. 2002.

[Gartner, 2012] Gartner Inc. *Predicts 2013: Application Integration*. 2012.

[Greenwood et al., 2007] Dominic Greenwood, Margaret Lyell, Ashok Mallya, and Hiroki Suguri, "The IEEE FIPA approach to integrating software agents and web services", in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems (AAMAS '07)*, pp 1412-1418, 2007.

[Harris, 2007] Robin Harris (2007). *SOA done right: the Amazon strategy* [Online]. Available: http://www.zdnet.com/article/soa-done-right-the-amazon-strategy/. [Accessed: December 2014].

[Hritz, 2012] Michael Hritz (2012). *SOA, Cloud and Services Technology In the FAA National Airspace System* [Online]. Available: http://www.infoq.com/presentations/SOA-Cloud-Services-FAA?utm_source=infoq&utm_medium=related_content_link&utm_campaign=relatedContent_presentations_clk. [Accessed: November 2014].

[Inden et al., 2013] U. Inden, N. Mehandjiev, L. Mönch, P. Vrba, "Towards an Ontology for Small Series Production", Mařík, V., Martinez Lastra, J. L., Skobelev P. (eds): *Industrial Applications of Holonic and Multi-Agent Systems*, Springer Verlag Berlin-Heidelberg, LNCS 8062, pp. 128-139, 2013.

[Kanungo et al., 2002] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A.Y. Wu, "An Efficient k-means Clustering Algorithm: Analysis and Implementation", in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.24, n.7, pp. 881-892, 2002.

[Kreger et al., 2009] H. Kreger, J. Estefan (2009). *Navigating the SOA Open Standards Landscape Around Architecture, Joint White Paper from OASIS, OMG, and The Open Group* [Online]. Available at: https://www.oasis-open.org/committees/download.php/32911/wp_soa_harmonize_ d1.pdf. [Accessed: November 2014].

[Lawson, 2011] Loraine Lawson (2011). *Data-driven Decisions a Best Practice, but Still Not Mainstream* [Online]. Available: http://www.itbusinessedge.com/cm/blogs/lawson/the-three-best-examples-of-successful-soas/?cs=16305. [Accessed: November 2014].

[Leitão, 2009] P. Leitão, "Agent-based Distributed Manufacturing Control: A State-of-the-art Survey", in *International Journal of Engineering Applications of Artificial Intelligence*, 22(7): 979-991, 2009.

[Leitão, 2011] P. Leitão, "A Holonic Disturbance Management Architecture for Flexible Manufacturing Systems", in *International Journal of Production Research*, vol. 49, n.5, pp 1269-1284, 2011.

[Leitão et al., 2013] P. Leitao, J. Barbosa, P. Vrba, P. Skobelev, A. Tsarev, D. Kazanskaia, "Multi-agent System Approach for the Strategic Planning in Ramp-up Production of Small Lots", in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp 4743-4748, 2013.

[Marín et al., 2013] C. A. Marín, L. Mönch, L. Liu, N. Mehandjiev, G. V. Lioudakis, D. Kazanskaia, V. Chepegin, "Application of Intelligent Service Bus in a Ramp-up Production Context", *CAiSE 2013*; June 17-21 2013, Valencia, Spain.

[Mendes et al., 2009] J. M. Mendes, P. Leitão, F. Restivo and A. W. Colombo, "Service-oriented Agents for Collaborative Industrial Automation and Production Systems", in *Proceedings of the 4th International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS'09)*, V. Marik, T. Strasser and A. Zoitl (eds), LNAI 5696, Springer, pp. 1-12, 2009.

[Mendes, 2011] J. M. Mendes, Engineering Framework for Service-oriented Automation Systems [Ph.D. Dissertation]. Faculdade de Engenharia da Universidade do Porto, Porto, 2011.

[OASIS, 2004] OASIS (2004). *UDDI Version 3.0.2* [Online]. Available at: http://www.uddi.org/pubs/uddi_v3.htm. [Accessed: November 2014].

[OASIS, 2006] OASIS (2006). *Reference Model for Service Oriented Architecture 1.0* [Online]. Available at: http://docs.oasis-open.org/soa-rm/v1.0/. [Accessed: November 2014].

[OASIS, 2007] OASIS (2007). *Web Services Business Process Execution Language Version 2.0* [Online]. Available at: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html. [Accessed: November 2012].

[OASIS, 2009] OASIS (2009). *Devices Profile for Web Services Version 1.1* [Online]. Available at: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.docx. [Accessed: November 2012].

[OASIS, 2012] OASIS (2012). Web Services Quality Factors Version 1.0 [Online]. Available at: http://docs.oasis-open.org/wsqm/WS-Quality-Factors/v1.0/WS-Quality-Factors-v1.0.doc. [Accessed: December 2012].

[OMG, 2012] Object Management Group (2012). *Service Oriented Architecture Modeling Language (SoaML), Version 1.0.1* [Online]. Available at: http://www.omg.org/spec/SoaML/1.0.1/ [Accessed: November 2012].

[Pautasso et al., 2008] C. Pautasso, O. Zimmermann, F. Leymann, "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision", in *Proceedings of the 17th international conference on World Wide Web (WWW '08)*, pp. 805-814, 2008.

[Pereira et al., 2013] A. Pereira, N. Rodrigues, J. Barbosa, P. Leitão, "Trust and Risk Management Towards Resilient Large-scale Cyber-Physical Systems", in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE'13)*, May 28-31, Taipei, Taiwan, 2013.

[Ribeiro et al., 2008] L. Ribeiro, J. Barata, P. Mendes, "MAS and SOA: Complementary Automation Paradigms", in *IFIP International Federation for Information Processing*, vol. 266, Springer Boston, pp. 259-268, 2008.

[Rodrigues, 2013] N. Rodrigues, "Towards Interoperability with Ontologies and Semantic Web Services in Manufacturing Domain", in *Doctoral Symposium in Informatics Engineering - DSIE'13*, pp. 129-137, 2013.

[Rodriguez, 2008] A. Rodriguez (2008). *RESTful Web services: The basics* [Online]. Available at: http://www.ibm.com/developerworks/library/ws-restful/. [Accessed: December 2014].

[Sarang, 2009] P. Sarang, *Practical Liferay: Java-based Portal Applications Development*. Apress, 2009.

[The Open Group, 2011a] The Open Group (2011). *Using TOGAF to Define and Govern Service-Oriented Architectures* [Online]. Available at: https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?publicationid=12390. [Accessed: November 2014].

[The Open Group, 2011b] The Open Group (2011). *The Open Group Service Integration Maturity Model (OSIMM), Version 2* [Online]. Available at: https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?publicationid=12450. [Accessed: November 2014].

[Vrba et al., 2014] P. Vrba, P. Kadera, M. Myslík, M. Klíma, "JBoss ESB Sniffer - Message Flow Visualization for Enterprise Service Bus", in *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE'14)*, 2014.

[W3C, 2004a] World Wide Web Consortium (2004). *Web Services Glossary* [Online]. Available at: http://www.w3.org/TR/ws-gloss/. [Accessed: November 2014]

[W3C, 2004b] World Wide Web Consortium (2004). *Web Services Architecture* [Online]. Available at: http://www.w3.org/TR/ws-arch/. [Accessed: November 2014]

[W3C, 2004c] World Wide Web Consortium (2004). *Web Service Management: Service Life Cycle* [Online]. Available at: http://www.w3.org/TR/wslc/. [Accessed: December 2014]

[W3C, 2005] World Wide Web Consortium (2005). *Web Services Choreography Description Language Version 1.0* [Online]. Available at: http://www.w3.org/TR/ws-cdl-10/. [Accessed: November 2014]

[W3C, 2007a] World Wide Web Consortium (2007). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)* [Online]. Available at: http://www.w3.org/TR/soap12-part1/. [Accessed: November 2014]

[W3C, 2007b] World Wide Web Consortium (2007). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language* [Online]. Available at: http://www.w3.org/TR/wsdl20. [Accessed: November 2014]

[W3C, 2007c] World Wide Web Consortium (2007). *Semantic Annotations for WSDL and XML Schema* [Online]. Available at: http://www.w3.org/TR/sawsdl/. [Accessed: November 2014]

[Wang et al., 2012] B. Wang, X. Zhou, G. Yang, Y. Lou, "Service Lifecycle Management in Distributed JBI Environment", Web Information Systems and Mining, Lecture Notes in Computer Science, Volume 7529, pp 431-438, 2012.

[Witten et al., 2011] H. Witten, Eibe Frank, and Mark A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques* (3rd ed.). Morgan Kaufmann Publishers, San Francisco, 2011.

[Wooldridge, 2002] M. Wooldridge, *An Introduction to Multi-Agent Systems*. John Wiley & Sons, 2002.

[Ziyaeva et al., 2008] G. Ziyaeva, Eunmi Choi, Dugki Min, "Content-Based Intelligent Routing and Message Processing in Enterprise Service Bus", in *International Conference on Convergence and Hybrid Information Technology, ICHIT '08*, pp.245-249, 2008.