
# THÈSE

## En vue de l'obtention du
## DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

**Délivré par :**
Institut National des Sciences Appliquées de Toulouse (INSA Toulouse)

**Discipline ou spécialité :**
Informatique et Réseaux

---

**Présentée et soutenue par**
Codé Diop
**le :** 29 Avril 2015

**Titre :**
Un Bus de Service Autonome pour les Systèmes Distribués à base de Services

An Autonomic Service Bus for Service-based Distributed Systems

---

**École doctorale :**
Mathématiques Informatique Télécommunications (MITT)

**Unité de recherche :**
LAAS-CNRS

**Directeurs de thèse :**
Ernesto Exposito
Christophe Chassot

**Rapporteurs :**
Sherali Zeadally
Romain Rouvoy

**Autres membres du jury :**
Mike Papazoglou
Jose Aguilar
Khalil Drira
Jean-Pierre Lorre

**Auteur :** Codé Diop

**Titre :** Un bus de service autonome pour les systèmes distribués à base de services

**Directeurs de Thèse :** Ernesto Exposito et Christophe Chassot

**Lieu et date de soutenance :** LAAS-CNRS, 29 Avril 2015

---

**Résumé**

Avec l'évolution des technologies de l'internet, les applications et plus généralement les systèmes distribués sont de plus en plus conçus en composant et interconnectant un ensemble de services distribués. Ces services pouvant être très hétérogènes, plusieurs approches et solutions pour la gestion de l'intégration et l'interopérabilité ont été proposées. De toutes ces propositions, les bus de services (ESB – Enterprise Service Bus) ont été désignés comme étant la solution la plus adaptée. Toutefois, le problème avec les ESB est qu'ils sont déployés dans un contexte très évolutif et très dynamique; un contexte dans lequel un grand nombre de services peuvent être fournis et utilisés de façon concurrente à travers le bus. L'utilisation concurrente de ces services mais aussi des ressources sous-jacentes allouées au bus (mémoire, processeur, etc.) peut conduire à des événements imprévisibles tels qu'une surcharge du bus, une indisponibilité des services, des temps de réponse élevés, une diminution de la fiabilité, etc. Dans ce contexte, des solutions efficaces permettant de garantir ou d'améliorer à la fois la qualité de service et l'évolutivité offertes par les ESB sont nécessaires.

Le but de cette thèse est de proposer les principes architecturaux pour la mise en place un bus de service autonome (ASB) qui offre une solution de communication scalable guidée par les transactions des systèmes interconnectés, mais aussi par les ressources disponibles. L'ASB offre aussi un service d'intégration différentiée en fonction des exigences en termes de qualité de service spécifiques aux systèmes interconnectés.

---

**Mots Clés :** Systèmes distribués, système de communication, architecture orientée service, bus de service, adaptabilité, qualité de service, intelligence artificielle

---

**Discipline :** Informatique et Réseaux

**Etre grand c’est soutenir une grande querelle**
Hamelet

**« Car comment serait-il possible, si le salut était là, à notre portée et qu'on pût le trouver sans grande peine, qu'il fut négligé par presque tous ? Mais tout ce qui est très précieux est aussi difficile que rare. »**
Spinoza

# Remerciements

Les travaux présentés dans ce manuscrit ont été effectués au sein du Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS) au sein du groupe Services et Architectures pour les Réseaux Avancés (SARA).

C'est avec une grande émotion que je voudrais remercier toutes les personnes ayant soutenu et apprécié mon travail.

Je tiens d'abord à remercier M. Jean ARLAT, Directeur du LAAS-CNRS pour son accueil et la qualité des moyens informatiques et logistiques mis à ma disposition. Je remercie tout particulièrement M. Khalil DRIRA, responsable du groupe SARA pour sa confiance et tout le soutien qu'il m'a apporté tout au long de ma thèse.

Je tiens à remercier très chaleureusement et à témoigner toute ma reconnaissance et ma gratitude à mes directeurs de thèse M. Ernesto EXPOSITO et M. Christophe CHASSOT pour m'avoir permis de réaliser ces travaux. J'ai reçu un encadrement solide et je les remercie pour les nombreuses réflexions que nous avons pu mener et au travers desquelles ils ont généreusement partagé leur savoir et expérience.

Je suis particulièrement reconnaissant envers M. Sherali ZEADALLY, Professeur à l'Université de Kentucky, et M. Romain ROUVOY, Maitre de Conférences à l'Université de Lille, qui m'ont fait l'honneur d'être les rapporteurs de cette thèse.

Je tiens également à remercier M. Jose AGUILAR, Professeur à l'Université Los Andes de Merida, M. Mike PAPAZOGLOU, Professeur à l'université de Tilburg, M. Jean Pierre LORRE, Directeur de l'innovation de Linagora et M. Khalil DRIRA, Directeur de recherche au CNRS, pour leur participation à mon jury de thèse.

Je remercie également tous mes collègues et amis du groupe SARA pour leur collaboration, leur soutien et leur bonne humeur (Guillaume, Lionel, M. Guillaume, Johan, Yassine, Mohamed, Ghada, Mahdi, Emna, Denis, Marc, Aymen, Cedric, Ismail, Rhiad, Nesrine, Ikbel, Tom) ainsi que les stagiaires qui ont eu à travailler avec moi durant cette thèse (Dalel, Roberto, Mariano, Serigne).

Je remercie également tous les membres du LAAS ainsi que le personnel administratif.

Merci à tous ceux qui ont su m'accompagner et me soutenir pendant ces années de thèse, je pense fortement à ma mère et mon père sans qui rien n'aurait été possible et à qui j'exprime ma profonde gratitude, ma femme chérie, mes frères et sœurs, mes tantes et oncles, ma (très) belle famille, mes amis qui font partie de la famille.

Enfin, je dédie ce travail, dans son intégralité, à mon frère Youssouf DIOP.

## Abstract

With the accelerated evolution of Internet, distributed systems are more and more designed as a composition of distributed services that need to be composed to implement complex business processes. Diversity and heterogeneity of these services raise important integrability and interoperability requirements. To meet these needs, the Enterprise Service Bus (ESB) has been proposed as a mediator based on open and standard interfaces facilitating the integrability and interoperability of services. However, in very active and dynamic contexts where a large number of concurrent services can be provided and consumed via the ESB, the competition for using shared services, but also the underlying computing resources allocated to the ESB (memory, processor, etc.) can lead to unpredictable events such as service unavailability, high response time, decrease of reliability, etc. Such anomalies need to be addressed by proposing efficient strategies able to guarantee or to improve both the QoS and scalability offered by the ESB.

The aim of this thesis is to propose an architectural framework for a QoS-aware Autonomic Service Bus (ASB) able to offer in an autonomic way a scalable communication solution guided by distributed systems transactions and the state of the underlying computing resources. The ASB offers also a differentiated integration service based on the QoS requirements of interconnected systems.

## Résumé

Avec l’évolution des technologies de l’internet, les applications et plus généralement les systèmes distribués sont de plus en plus conçus en composant et interconnectant un ensemble de services distribués. Ces services pouvant être très hétérogènes, plusieurs approches et solutions pour la gestion de l’intégration et l’interopérabilité ont été proposées. De toutes ces propositions, les bus de services (ESB – Enterprise Service Bus) ont été désignés comme étant la solution la plus adaptée. Toutefois, le problème avec les ESB est qu’ils sont déployés dans un contexte très évolutif et très dynamique; un contexte dans lequel un grand nombre de services peuvent être fournis et utilisés de façon concurrente à travers le bus. L’utilisation concurrente de ces services mais aussi des ressources sous-jacentes allouées au bus (mémoire, processeur, etc.) peut conduire à des événements imprévisibles tels qu’une surcharge du bus, une indisponibilité des services, des temps de réponse élevés, une diminution de la fiabilité, etc. Dans ce contexte, des solutions efficaces permettant de garantir ou d'améliorer à la fois la qualité de service et l'évolutivité offertes par les ESB sont nécessaires.

Le but de cette thèse est de proposer les principes architecturaux pour la mise en place un bus de service autonome (ASB) qui offre une solution de communication scalable guidée par les transactions des systèmes interconnectés, mais aussi par les ressources disponibles. L’ASB offre aussi un service d’intégration différentiée en fonction des exigences en termes de qualité de service spécifiques aux systèmes interconnectés.

# Author's publications

- Book: 1
- International journals: 5
- International conferences: 11
- Best Paper Awards: 3

**Book**

[1] Ernesto Exposito, **Codé Diop**, "Smart SOA platforms in cloud computing architectures", Wiley-ISTE, ISBN: 978-1-84821-584-9, June 2014, pp.224.

**International journals**

[2] Roberto Koh-Dzul, Mariano Vargas-Santiago, **Codé Diop**, Ernesto Exposito, Francisco Moo-Mena, Jorge R. Gomez-Montalvo: Improving ESB Capabilities through Diagnosis Based on Bayesian Networks and Machine Learning. JSW 9(8): 2206-2211 (2014)

[3] Mohamed Zouari, **Codé Diop**, Ernesto Exposito, "Multilevel and Coordinated Self-management in Autonomic Systems based on Service Bus", Journal of Universal Computer Science (JUCS), 20 (3), 2014, 431-460.

[4] Ghada Gharbi, Mahdi Ben Alaya, **Codé Diop**, Ernesto Exposito, AODA: an Autonomic and Ontology-Driven Architecture for service-oriented and event-driven systems, Int. J. Collaborative Enterprise, 3 (2/3), 2013.

[5] Amina Chaabane, **Codé Diop**, Wassef Louati, Mohamed Jmaiel, Jorge Gomez Montalvo, Ernesto Exposito, Towards a semantic-driven and scalable publish/subscribe framework, International Journal of Internet Protocol Technology, 1 (7), 2013.

[6] **Codé Diop**, Guillaume Dugué, Christophe Chassot, Ernesto Exposito, Jorge Gomez, QoS-aware and Autonomic-oriented Multi-Path TCP extensions for mobile and multimedia applications, International Journal of Pervasive Computing and Communications, 8 (4), 2012.

**International conferences with committee and proceedings**

[7] **Codé Diop**, Aymen Kamoun, Emna Mezgani, Ernesto Exposito, "A smart platform for dynamic enterprises collaboration", 20th European Concurrent Engineering Conference, April 28-30, 2014, Bruges, belgium. 8p.

*The paper received the Best Paper Award of ECEC '14*

[8] Roberto KOH DZUL, Mariano Vargas-Santiago, **Codé Diop**, Ernesto Exposito, Franscisco Moo-Mena, "A smart diagnostic model for an autonomic service bus based on a probabilistic reasoning approach", 10th IEEE International Conference on Autonomic and Trusted Computing (UIC/ATC 2013), Vietri sul Mare, Italia, 18-21 December 2013, pp.416-421.

[9] **Codé Diop**, Ernesto Exposito, Christophe Chassot. “QoS and scalability management in an autonomic cloud-based networked service bus”, 20th International Conference on Telecommunications, Casablanca, Morocco, 6-8 May 2013, 5p.

[10] **Codé Diop**, Emna Mezghani, Ernesto Exposito, Christophe Chassot, Khalil Drira. “QoS-driven Autonomic Abilities through a Multi-homed Transport Protocol”. 27th IEEE International Conference on Advanced Information Networking and Applications AINA 2013, Barcelona, Spain, mars 2013, 8p. 645-652.

[11] **Codé Diop**, Ernesto Exposito, Christophe Chassot, Dalel Jlidi. “QoS-aware and ontology-driven autonomic service bus”, IEEE International Conference on Collaboration Technologies and Infrastructures (IEEE WETICE 2012), Track on Management of Dynamic Networked Enterprises MADYNE, Toulouse (France), 25-27 Juin 2012.

[12] Gadha Gharbi, Mahdi Ben Alaya, **Codé Diop**, Ernesto Exposito. “AODA: an autonomic and ontology-driven architecture for service-oriented and event-driven systems”, IEEE International Conference on Collaboration Technologies and Infrastructures (IEEE WETICE 2012), Track on Adaptive and Reconfigurable Service-oriented and component-based Applications and Architectures AROSA, Toulouse (France), 25-27 Juin 2012.

[13] **Codé Diop**, Jorge Gomez-Montalvo, Guillaume Dugué, Christophe Chassot, and Ernesto Exposito. “Towards a semantic and MPTCP-based Autonomic Transport Protocol for Mobile and Multimedia Applications”. IEEE Third International Conference on Multimedia Computing and Systems (ICMCS 2012) May 10-12, Tangier, Morocco, 2012.

[14] Guillaume Dugué, **Codé Diop**, Christophe Chassot, and Ernesto Exposito. “Towards Autonomic Multipath Transport for Infotainment-like Systems”. IEEE International Conference on Communications ICC 2012, Ottawa (Canada), June 10-15, 2012.

*The paper received the Best Paper Award of IEEE ICC '12 / SACONET workshop*

[15] **Codé Diop**, Ernesto Exposito, Khalil Drira and Christophe Chassot. “Semantic-driven Autonomic Service Bus”. 6th International Conference of Interoperability for Enterprise Systems and Applications (I-ESA’12) - Workshop Factories of the Future (FoF), Valencia, Spain, March 2012.

[16] **Codé Diop**, Guillaume Dugué, Christophe Chassot, and Ernesto Exposito. “QoS-oriented MPTCP extensions for multimedia multi-homed systems”. International Workshop on Protocols and Applications with Multi-Homing Support (PAMS 2012), Fukuoka (Japon), 26-29 Mars 2012, 6p.

[17] **Codé Diop**, Guillaume Dugué, Christophe Chassot, and Ernesto Exposito. 2011. QoS-aware multipath-TCP extensions for mobile and multimedia applications. In *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia* (MoMM '11). ACM, Ho chi minh, Vietnam, 139-146. DOI=10.1145/2095697.2095723 http://doi.acm.org/10.1145/2095697.2095723

*The paper received the Best Paper Award of MoMM '11*

# Contents

# Chapter 1. **General Introduction**

**Contents**



Distributed systems have considerably evolved during the past years. In addition to the traditional Client-Server based systems, more complex distributed systems characterized by the multi-tiered design model are increasingly deployed within an organisation or across several interconnected enterprises.

With this evolution, several communication requirements have been identified such as Quality of Service (QoS), scalability, integrability, interoperability, etc. The dynamic context of distributed systems raise also requirements related to the manageability and self-manageability of the communication solutions.

To tackle these requirements, solutions were proposed at the network and transport TCP/IP layers of the communication stack. These solutions deal with QoS requirements of distributed systems. However, they present scalability issues or are not broadly deployed due to network constraints. Moreover, none of them was proposed to manage integrability and interoperability.

The communication middleware layer was introduced to deal with integrability and interoperability requirements of distributed systems. But existing communication middleware solutions can present QoS and scalability issues when a high number of concurrent transactions need to be supported.

In this thesis, a new communication middleware solution is proposed. Compared to existing solutions, our proposal deals with the integrability and interoperability, and also satisfies QoS and scalability requirements in an efficient and autonomic way.

The goals of this chapter are:

- to introduce the evolution of distributed systems and to identify the related requirements;
- to analyse the proposed solutions at the network, transport and middleware layers and to demonstrate their limits;
- to present and position our proposal to cope with these limits;
- and to introduce our different contributions.

The structure of the chapter is as follows: the distributed systems context is presented in section 1.1. The thesis problem is stated in section 1.2. The thesis positioning is developed in section 1.3. The different contributions are introduced in section 1.4. The structure of the dissertation is finally summarized in section 1.5.

### 1.1 Distributed systems context and requirements

During the last decades, Internet technologies have been highly developed, and in particular, there have been considerable evolutions in network technologies since the wired technologies offering low and limited bandwidth are more and more replaced by new ones (wired and wireless) providing high bandwidth (e.g. Ethernet, WiFi, 3G, 4G, etc.). Furthermore, current devices (e.g. PC, laptops, tablets, smart-phones, etc.) can now either be fixed or mobile, with divers characteristics (in terms of CPU, memory, etc.) and generally several network interfaces to better take advantage of the multiple network resources.

With this evolution, **distributed and multimedia systems** are more and more developed in addition to the **traditional file transfer or email applications**. Moreover, **complex distributed systems such as enterprise systems** based on the interconnection of a multitude of heterogeneous and distributed applications and data sources (e.g. Enterprise Resource Planning systems, Customer Relationship Management systems, Portal systems, etc.) are also present nowadays in the landscape of Internet distributed systems.

In this context, several protocols and services were proposed at the **network and transport OSI layers** to satisfy users requirements and to take advantage of new network technologies

(e.g. IntServ [BRA 94], DiffServ [NIC 98], MPLS [RFC 3031], DCCP [KOH 06], SCTP [STE 07], MPTCP [FOR 10], etc.). These protocols and services were proposed to better cope with QoS requirements, such as order and reliability for the traditional file transfer or email applications, or guaranteed bandwidth and bounded delay for video/audio streaming or conferencing multimedia applications.

The **communication middleware layer** was also developed with solutions designed to facilitate the interconnection of largely distributed systems (compared to conventional solutions based on TCP sockets), and especially to allow taking into account applications and data sources distribution and heterogeneity (e.g. Remote Procedure Calls [BIR 84], Distributed Objects [COM 13] [MIS 13] [RMI 13] [COR 13], Message Oriented Middleware [CUR 04], Event Driven Architecture [GAR 13], Resource Oriented Architecture [FIE 00] or Service Oriented Architecture [SOA 06]).

The communication middleware solutions and the current economic globalization context have also encouraged the development of new distributed systems resulting from the collaboration of multiple organizations by means of **inter-organizations systems based on the interconnection of their applications and systems**. The development of such distributed systems across enterprises or organizations is also promoted by the advent of the **cloud computing**, which provides reusable software, features and resources that can be easily consumed as a service.

The analysis of all these evolutions led us to identify three generations of distributed systems: a **first generation** based on the Client-Server design model characterized by two distributed entities deployed on distinct nodes interconnected via the network, a **second generation** based on the multi-tiered distributed model characterized by several tiers or components that communicate within an organization and a **third generation** based on the multi-tiered distributed model characterized by several tiers or components that communicate across several organizations.

Next sections detail these three generations of distributed systems.

### 1.1.1 First generation of distributed systems

The **first generation** is based on the Client-Server design model characterized by two distributed entities deployed on two distinct nodes interconnected via the network.

The Figure 1.1 illustrates the Client-Server design model. The server centralizes the business logic. The clients need to know the location and the identification of the server before using it. Clients and server usually operate in a synchronous way (e.g. clients pull data from the server by sending requests and waiting for responses) using specific APIs and languages implementation.

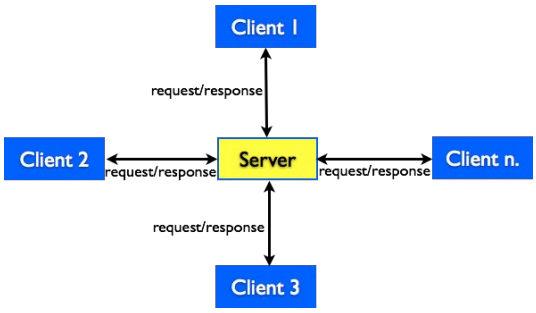


**Figure 1. 1 Client-Server design model**

The first generation of distributed systems is composed by traditional Internet applications (e.g. web servers, file transfer, Email), which mainly require a fully reliable and ordered data transfer. Another kind of applications such as multimedia and interactive applications (e.g. conferencing, streaming) has been also promoted with the broadband Internet deployment. These applications have more complex requirements since they process different kinds of multimedia data flows (e.g. audio and video).

Based on the characteristics of this first generation of distributed systems, requirements to be satisfied by the underlying communication layers have been identified and defined in [ITU 97] [EEL 05] [ITU 08] [IHR 13]. Among them, we consider in this thesis the following:

- ***Quality of Service*** defined as the ability of the communication solution to complete a transmission or a set of transmission tasks while respecting expected level of part or all of the following criteria:
  - ***reliability*** defined as the ability of the communication solution to complete successfully a transmission task. A reliable communication solution is able to recover messages that are lost, damaged, duplicated or received out of order;

  - o ***throughput*** defined as the communication solution capacity relative to the number of messages, events and/or data handled by the communication solution during a defined period;
  - o ***latency*** defined as the round-trip delay between the date when messages are sent and the date when responses are received;
  - o ***availability*** defined as the ability of the communication solution to stay in an operational condition. An available communication solution is able to ensure its role at each time it is invoked.
- ***scalability*** defined as the ability of the communication solution to handle a growing amount of work without degradation of the QoS. A scalable communication solution is able to guarantee the same level of QoS even if the number of transactions increases.

The complexity of the QoS and scalability management in the dynamic context of distributed systems (e.g. user mobility, traffic variation, etc.) justifies the introduction of a second set of requirements, which qualify how complex is the control of actions to manage QoS and scalability, and how the communication solution behaves in critical situations. These requirements are defined as follows [NOR 13]:

- ***manageability*** is the ability of the communication solution in operation to be monitored and controlled;
- ***self-manageability*** is the ability of the communication solution in operation to be self-monitored and self-controlled in order to adjust itself in dynamic environments.

Next section presents the characteristics of the second generation composed by more complex distributed systems.

### 1.1.2 Second generation of distributed systems

The **second generation** is based on the multi-tiered design model characterized by several distributed tiers or components that communicate generally within an enterprise or an organization.

In opposite to the Client-Server design model, in this generation, the applications are split in several tiers, components or entities (Figure 1.2):

- on the server side, the various applications functionalities are not centralized on an individual node. We have distributed components providing the presentation logic (Presentation-tier), the application logic (Business-tier) and enterprise information systems (EIS-tier);
- on the client side, we have the component providing user interactions capabilities (Client-tier).


**Figure 1. 2 Multi-tiered design model**

Accordingly, the applications that compose this second generation involve various distributed and interacting entities (tiers), which implement the business functionalities. They can be implemented using heterogeneous APIs and programing languages. Moreover, their operations involve the exchange of messages that enable the communication and the access to data, services, resources, events, etc. Exchanged messages can be in different formats (e.g. text, PDF, JPEG, HTML, JSON, etc.), and their transmission can be based on different protocols (e.g. SMTP, FTP, HTTP, etc.).

Consequently, in addition to the previous requirements of the first generation (QoS, scalability, manageability and self-manageability), systems of the second generation require from the underlying communication layers further abilities that can be expressed in terms of the following properties [IHR 13] [NOR 13]:

- ***integrability*** of the distributed entities: integrability is defined as the ability of a new entity to be easily aggregated to an existing distributed system. For instance, integrability means the ability to plug easily a new entity to add a new functionality to a system;
- ***interoperability*** of the distributed entities: interoperability is defined as the ability of a new entity to communicate with the existing entities of a distributed system. For instance, interoperability means the ability of existing entities of a distributed system to understand and exploit messages from an added entity.

New information communication technologies and the business opportunities of worldwide global markets have encouraged the development of a new generation of interconnected multi-tiered enterprise systems that will be presented in the next section.

### 1.1.3 Third generation of distributed systems

The **third generation** is based on the multi-tiered design model characterized by the composition and the interconnection of several distributed tiers or components that communicate across several enterprises or organizations (e.g. collaborative manufacturing networks of small and medium enterprises - SMEs).

The characteristics of the third generation imply several requirements to be satisfied by the underlying communication layers. Indeed:

- the systems can be largely distributed. They are present not only in one domain (intra-organization), but also in several domains (inter-organization);
- the systems components can be highly heterogeneous;
- the systems can present variable QoS and scalability requirements. These requirements can be different inside an organization, but also between several organizations;
- the different organizations can have various resources management protocols and goals, heterogeneous integration solutions and policies, etc.;
- the dynamicity of collaborative enterprises or organizations context can also make the satisfaction of requirements more complex [IMA 11].

Therefore the requirements are similar to the ones coming from the first and second generations. However, systems of the third generation evolve in environments where integrability, interoperability, QoS, scalability, manageability and self-manageability satisfaction is even more complex.

In this context, more efficient and advanced communication solutions are needed to support systems of this generation. In 2011, Gartner preconized the needs of cloud services integration as potential alternatives to traditional communication solutions (***Integration Platform as a Service - iPaaS***) [PER 11]. The emergence of these cloud-based integration solutions will simplify the development and the use of infrastructures and platforms intended to deploy and scale the resources required for interconnecting distributed systems within the same organization as well as across multiple organizations. Among the advantage of these solutions, we can also notice the externalization and the self-provisioning of resources that allow avoiding the complex management and the overprovisioning when the infrastructure is managed internally within the organizations.

### 1.1.4 Summary

Distributed systems have considerably evolved with the development of Internet technologies. Three generations of distributed systems have been identified and a set of requirements related to the underlying communication layers are considered in this thesis.

The first generation presents QoS and scalability requirements; and the manageability and self-manageability of the communication solution to satisfy these requirements depend on only two interacting entities. The satisfaction of these QoS, scalability, manageability and self-manageability requirements is more complex for the second generation, since they are more than two distributed entities. And in addition, integrability and interoperability need to be taken into account. And finally, the characteristics of the third generation (e.g. multi-domains, dynamicity of the context, etc.) make even more complex the satisfaction of QoS, scalability, integrability, interoperability, manageability and self-manageability requirements. The figure 1.3 and table 1.1 summarize characteristics of these three generations and their related requirements.

| Generations | Requirements | | | | | | Complexity of requirements management |
|---|---|---|---|---|---|---|---|
| | QoS | Scalability | Manageability | Self-manageability | Integrability | Interoperability | |
| 1 Client-Server applications or Peer-to-peer applications | X | X | X | X | | | * Between 2 entities |
| 2 Multi-tiered applications **within an organization** | X | X | X | X | X | X | *High heterogeneity and distribution of the entities (more than 2) |
| 3 Multi-tiered applications **across several organizations** | X | X | X | X | X | X | *Multi-organizations *Dynamicity of networked collaborative context |

**Table 1. 1 Distributed systems generations and communication requirements**

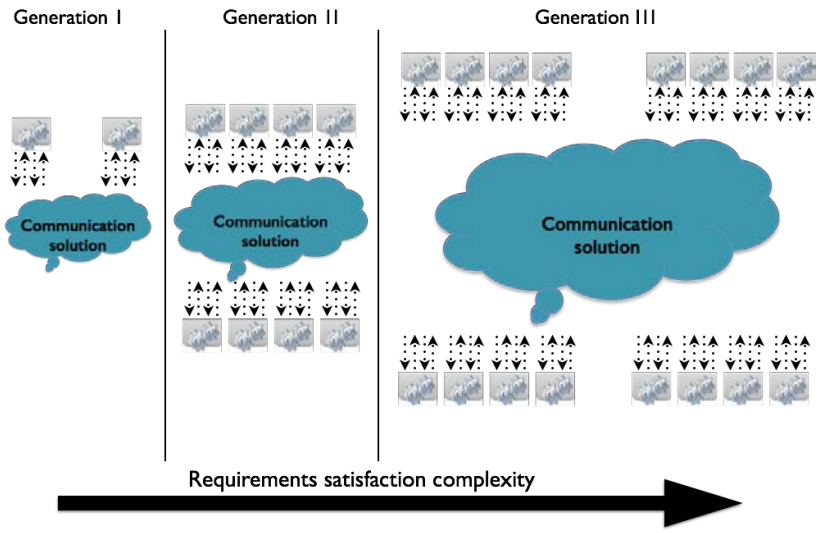


**Figure 1. 3 Client-Server and P2P applications over the network and transport services**

Several communication solutions have been proposed to deal with these requirements. These proposals are presented in the next section.

## 1.2 Existing communication solutions

In order to facilitate the development of distributed systems and to cover the requirements related to the underlying communication layers, evolutions and improvements were proposed at the different levels of network, transport and middleware layers. However these solutions partially cover the identified requirements. Network and transport layers proposals mainly address QoS requirements. Middleware layer proposals deal more with the integrability and interoperability requirements. The main goals of the following sections are to review and analyze the different communication solutions proposals considered in this thesis (based on distributed systems requirements).

### 1.2.1 Network and transport layers solutions

Several solutions were proposed at the network and transport layers to address QoS requirements.

QoS oriented models were proposed at the **network layer** to extend the traditional best-effort service model. The Integrated Services model (IntServ) [BRA 94] was recommended to manage the throughput and response time offered to applications. IntServ asks for an admission control to verify the availability of the communication medium. A reservation of resources is applied during the communication. These actions are based on the RSVP protocol [BRA 97]. The Differentiated Services model (DiffServ) [NIC 98] was promoted to offer different levels of QoS to concurrent applications. DiffServ is based on the configuration of network devices and resources to offer differentiated QoS levels to concurrent applications. The MultiProtocol Label Switching (MPLS) [RFC 3031] architecture has been proposed to specify a protocol improving the routing process of the Internet traffic flow.

The **transport layer** was also impacted with the development of distributed systems. In addition to the traditional and well known TCP (Transport Control Protocol) [POS 81] and UDP (User Datagram Protocol) [POS 80], several proposals have been conducted at the IETF (Internet Engineering Task Force) with the purpose of providing a better service to applications. Indeed, TCP provides a fully reliable and ordered service; and mechanisms to

ensure this service can include performance issues (i.e. high latency due to the retransmission or error control mechanisms). And UDP provides a best effort service without QoS and scalability guarantees. TCP and UDP services are not adapted to individually cope with the diversity of requirements of applications and sometimes they are suboptimal.

In this context, DCCP (Datagram Congestion Control Protocol) [KOH 06] was introduced as a protocol more suitable for applications that use UDP (e.g. non-reliable multimedia streaming applications) in the Internet since it includes congestion control to avoid network overload. SCTP (Stream Control Transport Protocol) [STE 07] was suggested as a protocol more suitable for contexts involving mobile users. SCTP takes into account the multiple network interfaces used by the current devices. In 2011, the IETF initiated the standardization of a new protocol called MPTCP (Multi-path Transport Control Protocol) [FOR 10] [FOR 11], which aims at replacing TCP in the coming years. MPTCP is a set of extensions of TCP allowing the use of multiple paths between two hosts offered by the new generations of devices, whether mobile or not (e.g. PC, smartphone, tablets, etc.). MPTCP mechanisms aim at enhancing the use of network resources and the performance provided to applications.

The important number of solutions that were proposed at the network and transport layers mainly addresses QoS requirements. However, most of these proposals are not broadly deployed over the Internet due to their limits (in terms of scalability) or due to network constraints (more than 90% of Internet traffic is transported via TCP [WOL 07]). For instance, the IntServ QoS model does not scale when the network size grows since routers have to maintain the state of flows [YAN 04]. Nowadays, network operators mainly deploy DiffServ, IntServ and MPLS solutions in order to improve the QoS and to perform prioritization and differentiation between their different clients. However, distributed systems cannot directly exploit them, since they do not access an API for expressing their requirements, for reserving resources or configuring differentiation policies. Also, new proposed transport protocols are not largely deployed because of the presence of middleboxes (NATs, firewalls, proxies…) in the network, which can recognize and allow more TCP packets [CAR 02].

The diversity of proposed transports protocols has also made complex the smooth development (in term of integrability) of distributed systems (Figure 1.4). For instance, for entities of traditional Client-Server or Peer-to-Peer based applications, which interact using directly the transport layer API, the choice of services at transport layer is hard coded during

the development. To deal with this complexity of choice, developers working directly over the transport layer application-programming interface (API) need a high level of expertise on all the services offered by the large set of transport protocols. Moreover, they must have a good understanding of how these transport services can be combined with the underlying services offered at the network level.

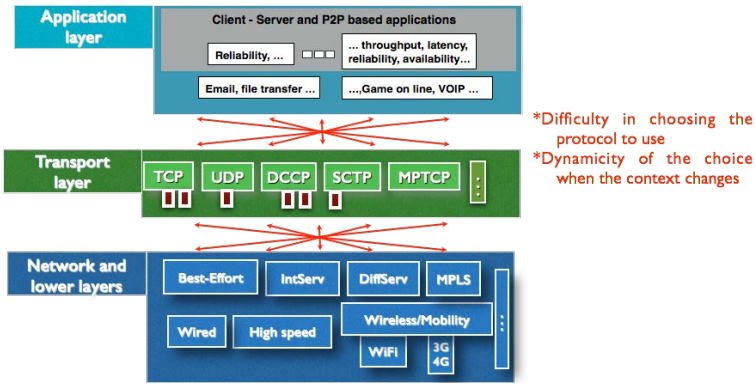


**Figure 1. 4 Client-Server and P2P applications over the network and transport services**

Also, once hard coded, none of these protocols allows taking into account the potential evolution of applications QoS requirements (e.g. evolution of multimedia session in terms of the number of multimedia flows or transmission requirements) or changes in network characteristics (e.g. decrease of the capacity or available bandwidth that can happen with users' mobility). In this context, more sophisticated transport protocols have to be developed to face these different issues and deal with the manageability and self-manageability requirements by taking into account dynamically and autonomously both applications requirements evolutions and network resources variations [EXP 12] [DIO 13] [WAM 13].

Let us finally note that existing network and transport protocols have not been proposed to deal with the integrability and interoperability required by distributed systems (mainly second and third generations).

Next section introduces the communication middleware layer proposed for the management of integrability and interoperability requirements.

### 1.2.2 Communication middleware solutions

Several **communication middleware solutions** were proposed to facilitate the development of distributed systems and mainly deal with the integrability and interoperability required by (second and third generations of) distributed systems.

The communication middleware layer is intended to hide all the complexity related to the distribution and heterogeneity of applications components, but also to hide the complexity induced by the evolution of the network and transport layers [KRA 04] [PIN 04]. It facilitates the integrability of distributed entities that should interact as they were located on the same execution environment, and provides interoperability functionalities to manage the heterogeneity of components by providing uniform and standard interfaces and making abstraction of the implementation languages (e.g. Java EE, .NET, C and C++).

Solutions such as Remote Procedure Calls, Distributed Objects, Message Oriented Middleware, Event Driven Architecture, Resource Oriented Architecture or Service Oriented Architecture were developed over the network and transport layers (Figure 1.5).

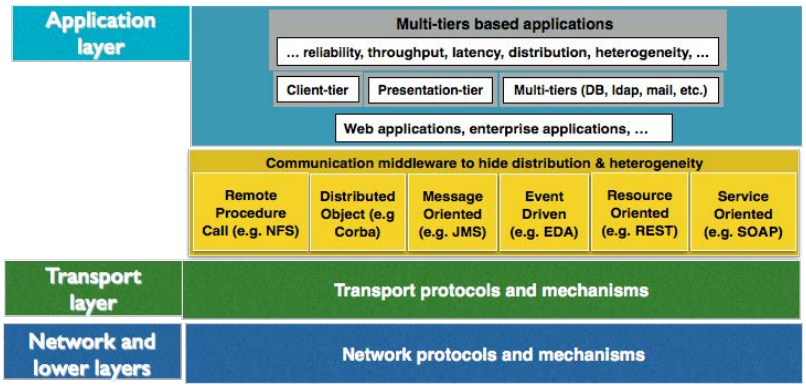


**Figure 1. 5 Middleware solutions over the network and transport layers**

The next paragraphs review and analyze the different middleware solutions regarding integrability, interoperability, QoS and scalability properties.

The Remote Procedure Call (RPC) [BIR 84] approach is mainly intended to deal with the integrability. It was proposed to reduce the complexity of directly using transport and network services and to allow an entity to invoke a remote procedure, as it was local. RPC promotes limited interoperability support. It does not integrate adaptation and mediation solutions and the entities interact using specific APIs and languages implementations. Interface Definition Language (IDL) is used to allow communication between heterogeneous entities [PIN 04].

The use of the RPC approach limits the distribution (the design is closer to the Client-Server model with synchronous request/reply interactions), the offered QoS (only based on the used transport and network protocols) and scalability (since a server centralizes the business logic) [TAL 00].

The Object Oriented (OO) [COM 13] [MIS 13] [RMI 13] [COR 13] approach extends RPC and is mainly intended to deal with the integrability. It is based on Object Request Brokers (ORB) introduced to hide the complex use of transport and network services by allowing an entity to interact with an object as it was in the same memory space. Even if it exists ORB solutions such as CORBA that offers interoperability functionalities to allow distributed objects written in multiple languages to interact, the OO approach promotes limited interoperability support and does not integrate adaptation and mediation solutions. The ORB solutions follow the Client-Server design model and support synchronous communications. Basically, they lack scalability and do not address QoS requirements [TAL 00]. Advanced ORB solutions supporting both synchronous and asynchronous communications, (e.g. CORBA v3) and dealing with QoS (e.g. Fault-Tolerant CORBA to deal with reliability) was proposed [EMM 00].

The Message Oriented [AMQ 12] and Event Driven [MIC 06] approaches give integrability solutions to develop more flexible and distributed systems based on asynchronous messaging solutions. The Message Oriented Middleware (MOM) was developed to support communications between several distributed entities, which interact by producing and consuming messages. The Event Driven Architecture (EDA) was proposed to implement middleware solutions similar to MOM, but the interconnected entities are more reactive or proactive as they publish and subscribe to specific kind of messages defined as events. These middleware solutions implementing the message-oriented paradigm satisfy integrability requirements by supporting both synchronous and asynchronous communications. However, interoperability management is their limit. For instance, generally they impose specific APIs as JMS (Java Message Service) and do not integrate adaptation and mediation solutions. To deal with this limit and to give solutions to deal with the interoperability, protocols such as AMQP (Advanced Message Queuing Protocol) [AMQ 12] are more and more proposed. Message oriented solutions offer mechanisms to deal with the QoS (e.g. data persistency,

error recovery, prioritization of message, etc.). However, they have limited scalability and issues can occur if several transactions need to be managed [PIN 04].

The Resource Oriented Architecture (ROA) [FIE 00] was introduced to integrate distributed entities that expose their functionalities and data as web resources. The ROA approach based on the REST architectural style and the HTTP protocol deals with the interoperability of distributed systems. However, it allows building Client-Server based applications that present limitations in terms of large integrability, distribution and scalability [TAL 00]. Also, without the web (HTTP protocol), REST becomes irrelevant since it does not support exchanges based on other synchronous or asynchronous communication protocols such as JMS, SMTP, FTP, etc. [FOU 08].

The Service Oriented Architecture (SOA) [SOA 06] was also proposed for supporting integrability between distributed entities that share features and data as services. The Service Oriented Architecture also guarantees the interoperability as it promotes complete distributed solutions based on standard communication protocols and mechanisms to integrate various and heterogeneous platforms. SOA offers an approach to build systems as a set of independent heterogeneous and distributed services available somewhere in the network and able to exchange data. A service bus is used to provide the required integration functionalities between service providers and service consumers; and to allow them to interact and to become interoperable. Several technological frameworks and solutions have been proposed to support the SOA paradigm and manage the integrability and interoperability of services such as:

- Application Server (AS) solutions that provide central services containers performing as runtime environments for hosting and running software components. These service containers are able to manage all the interactions with the components but also the components lifecycle, transactions, resource allocation, and security. AS solutions deal with the integrability problem but not necessarily with the interoperability since service consumers and providers need to manage themselves the adaptation of heterogeneous data format and protocols. Being a centralized solution, AS solutions have QoS and scalability problems.
- Message Oriented Middleware (MOM) solutions that offer asynchronous communication between message producers and consumers. A MOM is based on message and channel concepts and it is a good solution to satisfy integrability requirements. However it does not

satisfy interoperability requirements since service consumers and providers need to be adapted to use the MOM solution format and protocols. As already said, message oriented solutions offer mechanisms to deal with the QoS (e.g. data persistency, error recovery, prioritization of message, etc.). However, they can have scalability issues when several message channels and transactions need to be managed [PIN 04].

- Enterprise Application Integration (EAI) solutions that are aimed at offering integrability support services between enterprise applications. The basis of the EAI solutions implementation is a message broker acting as a central hub and offering a common API for application integration. Adapters are used to manage the interoperability. However, these adapters need to be implemented for each couple of heterogeneous consumers and providers, and their management can be hard. Also, service consumers and providers need to be adapted to use the specific offered API offered by the EAI solutions. EAI solutions have QoS and scalability problems resulting from bottlenecks and faults of the central hub.
- Enterprise Service Bus (ESB) solutions that have been proposed, in contrast with centralized EAI solutions, as a more efficient approach to ensure integrability and interoperability between distributed and heterogeneous components in a Service Oriented Architecture [CHA 04]. An ESB implements standard-based and distributed integration and mediation strategies such as discovery and invocation of services, composition of services, transformation of messages, message routing, protocol bridging, etc., allowing various and heterogeneous providers and consumers to share services, events, messages and resources in a synchronous or asynchronous way [ORT 07]. In contrast with centralized EAI solutions, ESB solutions improve the QoS and scalability since their components can be centralized on a single computer or distributed across multiple interconnected computers by federating several ESB instances. However they can present issues when they have to support a high number of transactions.

The figure 1.6 summarizes and classifies the integration efforts in SOA architecture [CHA 04]. The ESB is identified as the key element to support the SOA paradigm. It gives the most efficient communication technology to integrate any kind of heterogeneous and distributed services and processes. The ESB is also capable of promoting integrability and interoperability between heterogeneous middleware systems (Figure 1.7).

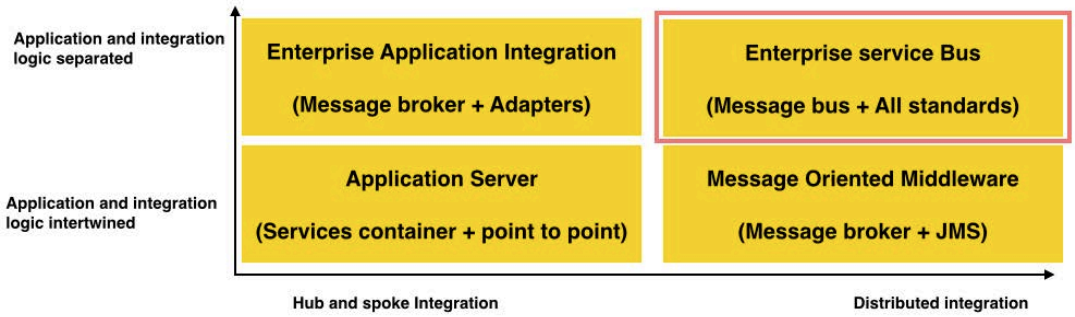


**Figure 1. 6 Services integration and mediation solutions**

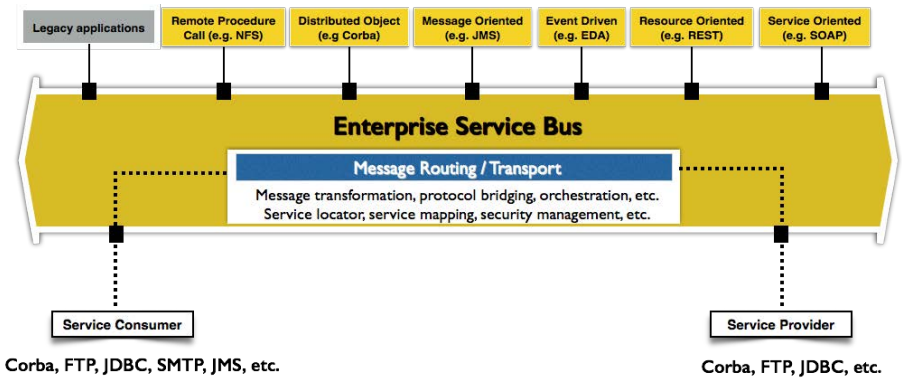


**Figure 1. 7 ESB functionalities**

Next section summarizes the study of existing communication solutions to state the focus of the thesis's contributions.

### 1.2.3 Summary

In order to facilitate the development of distributed systems and to satisfy their requirements related to the underlying communication layers, solutions were proposed at the different levels of network, transport and middleware layers. Several conclusions may be raised from the study of these solutions:

- an important number of solutions were proposed at the network and transport layers to address QoS requirements, but these proposals are not broadly deployed;
- proposals at the network and transport layers were not made for integrability and interoperability requirements;
- communication middleware solutions were introduced to mainly deal with integrability and interoperability requirements;
- SOA paradigm can be considered as the communication middleware solution able to better satisfy the integrability and interoperability requirements of distributed systems;
- within the SOA, the ESB is identified as the most efficient communication technology to integrate any kind of heterogeneous and distributed services and processes.

- Based on these conclusions, the ESB can be chosen as the key element to support a proposal aimed at satisfying requirements of distributed systems.

However, although ESB solutions allow integrating and making interoperable heterogeneous and distributed systems, they can present QoS and scalability issues when they have to support a high number of transactions. Indeed, the ESB is composed of a set of containers that host engines implementing the different integration and mediation functionalities. For these reasons, the ESB could represent a bottleneck and get congested or saturated, when a high number of transactions need to be exchanged. Congestion could appear due to the number of requests to be processed, but also due to the fact that the ESB runs on IT infrastructures with limited resources (CPU/threads, memory, network resources, etc.).

Nowadays, applied solutions to satisfy the QoS and scalability of ESB solutions are based on redundant topology models (e.g. clustering and load balancing on several instances) and overprovisioning techniques [CAL 08] [SIL 11] [WSO 11] [PAN 12]. And for instance using the overprovisioning techniques is a problem since allocated resources are not efficiently used. Mechanisms were also proposed to extend ESB implementation in order to improve for instance the message routing process that is an important task of an ESB [WU 09] [ZHO 10] [WU 10]. However, the proposals have limits since they are exclusively intended either to the QoS management or to the scalability management and not to both at the same time. And the proposals for the QoS management only deal with part of the identified QoS parameters (e.g. only the reliability or only the latency) and can create other scalability and QoS issues (e.g. the replica solution proposed in [WU 09]). To finish, let us state that existing proposals are mainly based on an opaque and extended ESB with hard-wired mechanisms configured statically without manageability and self-manageability properties required in the dynamic context of distributed systems.

Therefore, solutions are still needed to consider the autonomic and smart management of the QoS and scalability requirements of ESB. These solutions are among the main challenges for the middleware layer in the Future Internet due to the increasing number of networked systems (100 billions terminal in 2015) and the volume of exchanged data (42070 exabytes) [ISS 11].

Next section presents our position to deal with these challenges.

### 1.3 Thesis positioning

The state of art of the communication middleware solutions allows us stating that the SOA paradigm is the most adopted approach for the integrability and interoperability management of heterogeneous and distributed systems [THE 03] [CHE 06] [SOA 06] [EXP 14].

Moreover, the ESB is identified as the key element to support the SOA paradigm [CHA 04]. However, the ESB presents QoS and scalability issues when a high number of concurrent transactions need to be supported.

In this context, the main goals of this thesis are to enrich functionalities of existing ESB to offer a communication solution able to deal with integrability and interoperability, but also to satisfy in an autonomic and smart way (manageability and self-manageability) the QoS and scalability requirements of distributed systems.

Our **followed approaches** are:

**For QoS and scalability satisfaction,** we propose new mechanisms to extend the ESB solutions. Two complementary approaches have been explored:

- in the first one, the proposed mechanisms are inspired from our background in transport and network-oriented solutions for QoS management (e.g. congestion control, error control, shaping, differentiation, resource reservation, admission control, etc.). For instance, the congestion control mechanism is deployed within the ESB to avoid congestion of communication channels and to limit the usage of a saturated service provider. These mechanisms will be classified as intra-bus mechanisms;
- in the second approach, the proposed mechanisms are based on virtualization and cloud-computing characteristics (e.g. clustering, federation, load balancing, elasticity, self-provisioning, live migration, etc.) to avoid the overprovisioning or oversized techniques and to manage well resources of the IT infrastructures. For instance, the elasticity mechanism is applied on the underlying computing resources (e.g. processor, memory, storage resources, etc.) according to running transactions. These mechanisms will be classified as extra-bus mechanisms.

**For manageability and self-manageability satisfaction,** we propose an architecture that allows having a system able to manage in a dynamic and autonomic way the control and

coordination of proposed QoS and scalability oriented mechanisms. Indeed, limits of a static configuration in the current context of distributed systems motivate us to propose a dynamic solution. A dynamic solution can be manageable, being monitored and controlled by a human administrator. It can also be self-manageable with the ability to adjust itself, to be self-monitored and self-controlled. In this thesis, the proposed solution is both dynamic and self-manageable. It is based on the Autonomic Computing framework introduced by IBM [IBM 05].

Therefore, the architectural framework of the proposed ASB results in the extension of existing ESB solutions with QoS and scalability oriented mechanisms. Solutions implementing the autonomic computing framework are also included to ensure the control and coordination of these mechanisms in an autonomic way.

Several architectural design requirements come from the ASB proposal namely:

- a generic architecture to be implemented using any existing ESB implementations;
- a flexible and extensible architecture to allow the easy introduction of mechanisms for QoS and scalability requirements.

To satisfy these requirements, our **main architectural design foundations** are:

**For a generic architecture,** we adopt the Model Driven Architecture (MDA) [SOL 00] design methodology to have an open and generic solution that can be instantiated using any kind of ESB implementations;

**For a flexible and extensible architecture,** we adopt the Java Business Integrator (JBI) standard and specification proposed by the Java Community Process (JCP) in the JSR 208 specification [TEN 05] to have an open and extensible solution. JBI defines a platform for building ESBs using a set of plug and play services components. It allows integrating the proposed QoS and scalability mechanisms as a set of pluggable components.

The figure 1.8 summarizes the followed approaches and design foundations. The SOA paradigm (and ESB technology), network and transport oriented mechanisms, virtualization, cloud and autonomic computing are followed to build the ASB and cover identified requirements.

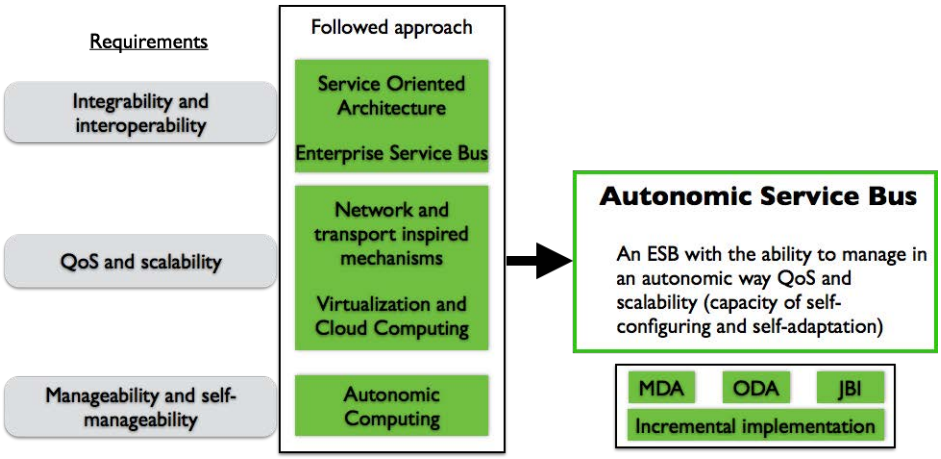


**Figure 1. 8 ASB proposition approach**

Next section introduces the thesis contributions.

### 1.4 Thesis contributions

**Our first scientific contribution** consists in the proposition of new mechanisms to extend ESB solutions for QoS and scalability management. The originality of these mechanisms is that they are inspired from classical and enhanced QoS-oriented mechanisms proposed within network and transport protocols and on virtualization and cloud computing paradigm for an efficient management of computing resources and scalability.

**Our second (and main) scientific contribution** deals with the architecture required to manage and coordinate the use of the proposed mechanisms to extend the ESB solutions for QoS and scalability management. By following the Autonomic Computing framework introduced by IBM, components and services are designed for an **Autonomic Service Bus (ASB)** able to manage in an autonomic way the QoS and scalability.

**Our third scientific contribution** consists in a specific implementation of the proposed architecture based on the OpenESB open source solution. An Emulation Platform for ESB Systems has also been developed and used during all the different phases of the implementation.

A deployment and evaluation of the ASB implementation have been done in the framework of the IMAGINE European project proposed in the domain of dynamic manufacturing network and Factories of Future [IMA 11]. IMAGINE research project targets the development and delivery of a new comprehensive methodology and a platform for effective end-to-end management of dynamic manufacturing networks. It illustrates well the second and third

generations of distributed systems since it allows distributed collaboration within extended enterprises or networked organizations. The complete development and validation have been done particularly through the integration of our solution in an aerospace and defence living lab environment.

## 1.5 Dissertation roadmap

The remainder of this dissertation is organized as follows:

Chapter 2 entitled “General background and related works” details the state of the art. It firstly presents the evolutions and improvements proposed at the different levels of network, transport and middleware layers. Secondly, research works that are more related to ESB solutions and how they have been improved are introduced.

Chapter 3 entitled “Autonomic Service Bus Architectural Framework” presents a generic architecture for building an ASB. The chapter gives an overview of how the ASB is structured and how it behaves. Proposed mechanisms for QoS and scalability management are introduced and the architectural design of the ASB is detailed. The design of the platform to go towards the ASB implementation based on Java Business Integration open standard is presented.

The chapter 4 entitled “Monitoring” presents a monitoring solution for detecting QoS and scalability issues. The solution is based on monitoring services proposed to supervise the different elements that need to be observed and an event-processing module proposed for the treatment of all the monitored data and the detection of symptoms to correct.

The chapter 5 entitled “Analysis, Plan and Execution” presents solutions developed to build a models for the analysis of monitored data and symptoms, and for the definition of the adequate plan to be executed. A first section presents the development of extensible models for a reliable and smart analysis of symptoms. The models are designed and used to diagnose symptoms. A second section presents the development of models and rule-based systems that guide the choice of the adaptive plans to deploy when anomalies are detected. The models are used to correlate a diagnostic to a corrective solution. The final part of this chapter details the execution services aimed at automatizing the use and coordination of mechanisms proposed in order to satisfy the scalability and the QoS demands.

Chapter 6 entitled “Dissemination, application and evaluation” introduces the IMAGINE European project [IMA 11] and presents how we evaluate services proposed for an implementation of the ASB based on IMAGINE scenarios.

Conclusions and perspectives of this thesis are finally presented in Chapter 7.

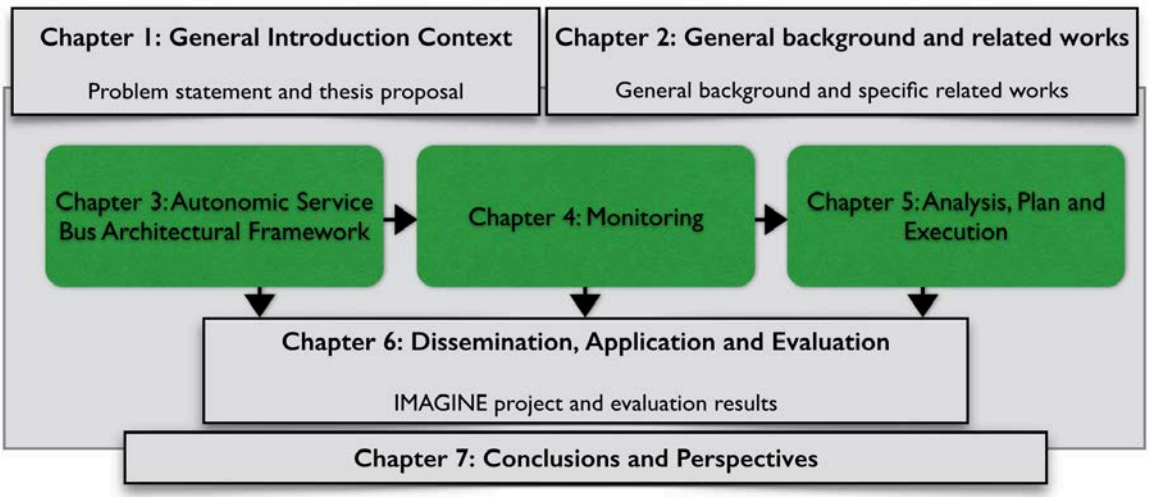


**Figure 1. 9 Dissertation structure**

# Chapter 2. **General background and related works**

**Contents**



## 2.1 Introduction

Distributed systems have considerably evolved during the past years. With this evolution, three generations of distributed systems and several requirements related to the underlying communication solutions (i.e. QoS, scalability, integrability, interoperability, manageability and self-manageability) have been identified in Chapter 1.

To tackle these requirements, several protocols and services have been proposed at the different levels of the network and transport OSI layers (e.g. IntServ [BRA 94], DiffServ [NIC 98], MPLS [RFC 3031], DCCP [KOH 06], SCTP [STE 07], MPTCP [FOR 10], etc.). Communication middleware solutions have also been introduced (e.g. Remote Procedure Call [RFC 1057] [RFC 5531], Object Oriented [COM 13] [COR 13] [MIS 13] [RMI 13], Message Oriented Middleware [AMQ 11], Event Driven Architecture [MIC 06], Resource Oriented Architecture [FIE 00] and Service Oriented Architecture [SOA 06]).

In this chapter, these network and transport layers proposals are detailed as the general background of this thesis. The main goal of this thesis being to enhance the communication middleware solutions, their evolution is also presented in this chapter. We present how they deal with integrability, interoperability, QoS and scalability requirements.

A focus is done on the Service Oriented Architecture (SOA) considered as the most efficient paradigm to guarantee integrability and interoperability requirements of distributed and heterogeneous systems [THE 03] [CHE 06] [SOA 06] [EXP 14].

We also present research works more specific to our thesis proposal. In particular, these works show how the Enterprise Service Bus (ESB) technologies identified as the key element that gives the most efficient technical solution to deploy SOA-based systems have been enhanced to deal with QoS, scalability, manageability and self-manageability.

The study of these specific research works allows justifying the proposal of the Autonomic Service Bus (ASB) as a new ESB able to manage in an autonomic and smart way QoS and scalability requirements of distributed systems.

The chapter is structured as follows: section 2.2 introduces network, transport and middleware layers proposals to deal with distributed systems requirements. Section 2.3 focuses on proposals enhancing ESB features. Section 2.4 concludes the chapter.

## 2.2 General background

In order to facilitate the development of distributed systems and to cover the requirements related to the underlying communication layers, evolutions and improvements were proposed at the different levels of network, transport and middleware layers. The following sections review and analyze the different proposals.

### 2.2.1 Network and transport layers solutions

Evolutions and solutions were proposed to enhance the network and transport OSI layers to mainly deal with QoS requirements. These classical and enhanced mechanisms proposed within network and transport protocols solutions are not broadly deployed nowadays due to scalability issues or to network constraints. However, they have inspired part of mechanisms proposed in this thesis to manage QoS and scalability requirements at the middleware layer. The following paragraphs present the most relevant.

#### 2.2.1.1 Evolution at the network layer

Several defined QoS-oriented models have been proposed at the network layer to extend the traditional best-effort Internet service model (at the IP level) characterized by a delivery of data packets without any guarantee on loss/error rate, order, throughput, delay and jitter.

*IntServ*

The Integrated Services model (IntServ) [BRA 94] has been recommended to guarantee the throughput and response time offered to applications. IntServ asks for an admission control to verify the availability of the bandwidth along the data path. A reservation of resources is applied before and during the communication. These actions are based on the signalling named Resource Reservation Protocol (RSVP) [BRA 97]. The IntServ model, in addition to the best effort service, proposes two services classes namely:

- the guaranteed service, providing bandwidth and delay guarantees;
- the controlled-load service, aimed at providing a QoS similar to the one offered by a network used in “normal” load conditions.

*DiffServ*

The Differentiated Services (DiffServ) [NIC 98] model has been proposed to offer different levels of reliability and performance to concurrent applications and systems. DiffServ is based on network devices and resources configurations to offer differentiated QoS performance to the concurrent applications data flows. The DiffServ model proposes two kinds of classes of services (or per hop behaviour):

- expedited forwarding (EF) with low delay and low jitter guarantees;
- assured forwarding (AF) with different service classes (four) and performance guarantees.

#### 2.2.1.2 Evolution at the transport layer

The transport layer has also been impacted with the development of distributed systems. In addition to the traditional and well known TCP (Transport Control Protocol) [POS 81] and UDP (User Datagram Protocol) [POS 80], several proposals have been conducted at the IETF (Internet Engineering Task Force) with the purpose of providing a better service to applications. Indeed, TCP provides a fully reliable and ordered service, and mechanisms to ensure this service can include performance issues (i.e. high latency due to the retransmission or error control mechanisms). UDP provides a best effort service without QoS guarantees. TCP and UDP services are sometimes suboptimal, and so not adapted to the diversity of requirements of distributed systems.

The following paragraphs introduce main standardized protocols and proposed extensions that have been conducted at the IETF (Internet Engineering Task Force). Among the multitude of transport level solutions that have been proposed in the literature, three of them have been standardized: DCCP, SCTP and MPTCP.

*DCCP*

The Datagram Congestion Control Protocol (DCCP) [KOH 06] has been introduced as a protocol more suitable for applications that use UDP in the Internet context. It delivers a service that includes a configurable congestion control (e.g. TCP's one or another one better adapted to the application's features) to avoid network overload induced by UDP (which has no congestion control), and to improve the efficiency of TCP with regard to (particularly multimedia) applications QoS requirements.

*SCTP*

The Stream Control Transport Protocol (SCTP) [STE 07] is connection-oriented, unicast, session-oriented, datagram-oriented and offers a reliable service. It can manage several separated data streams with the aim to take advantage of the "multi-homing" capability of existing devices. This capability consists in the fact that most of end hosts are connectable to the Internet with several interfaces and thus several IP addresses. SCTP was initially deployed for transporting signalling messages for Voice over IP (VoIP). It is now proposed for other uses and applications, among them mobile and multimedia applications.

Despite being implemented in several operating systems, SCTP is not used much due to the drawbacks it has from an application point of view (developers have to modify implementation of their applications to use SCTP), and because of the presence of middleboxes (NATs, firewalls, proxies, etc.) in the network, which can recognize and allow TCP packets (some can recognize UDP too), but block SCTP packets. In this context, the IETF recently initiated the standardization of a new protocol, MPTCP [FOR 10] [FOR 11], which aims at replacing TCP in the coming years.

*MPTCP*

The Multi-Path Transmission Control Protocol (MPTCP) is a set of extensions for legacy TCP that are transparent for both the applications and the network. This protocol is connection-oriented and allows the concurrent utilization of current hosts' multiple interfaces

and access technologies (“multi-homing”) to improve QoS on parameters such as sending rate or delay [FOR 10] [FOR 11].

Several works intended to extend these standard protocols were also proposed to deal with QoS and scalability [CON 94] [CON 96] [ROJ 99] [Wu 00] [ZHA 01] [ITO 02] [STE 04] [DIO 11], [DRE 11] [DIO 12], but also with manageability and self-manageability [EXP 10] [EXP 12] [WAM 13] [DIO 13].

To conclude the network and transport layers proposals section, let us state that several solutions were proposed; however, the evolution and diversity of introduced solutions have made complex the development of distributed systems. Today, developers working directly over the transport layer application-programming interface (API) need a high level of expertise on all the services offered by the large set of transport protocols. Moreover, they must have a good understanding of how those transport services can be combined with the underlying services offered at the network level.

Additionally, most of these proposals are not broadly deployed over the Internet due to their limits (in term of scalability) or due to network constraints (more than 90% of Internet traffic is transported via TCP [WOL 07]). For instance:

- the IntServ QoS model does not scale when the network size grows since routers have to maintain the state of flows [YAN 04]. Nowadays, network operators mainly deploy DiffServ and MPLS solutions in order to improve the QoS and to perform prioritization and differentiation between their different clients. However, distributed systems cannot directly exploit them, since there is not an easy way for them to access an API for expressing their requirements, for reserving resources or configuring differentiation policies;
- also, new proposed transport protocols are not largely deployed because of the presence of middleboxes (NATs, firewalls, proxies…) in the network, which can recognize and allow more TCP packets [CAR 02].

Finally, let us note that, by definition of the network and transport layers, none of these services is proposed to deal with the integrability and interoperability requirements induced by the second and third generations of distributed systems.

Next section introduces middleware layer solutions intended to deal with integrability and interoperability.

### 2.2.2 Evolution of communication middleware solutions

The communication middleware frameworks were introduced several years ago in order to facilitate the development of distributed systems, by providing a software layer to manage the distribution and the heterogeneity [KRA 04].

The main goal of a communication middleware is to simplify the use of transport and network layers services by:

- hiding distribution: developers do not need to be aware of the local or remote location of the system components;
- hiding heterogeneity: developers do not need to be aware of the language or platform-specific nature of the components in order to allow their inter-operation;
- providing uniform and standard-based interfaces to achieve easily integration of components;
- providing generic reusable components/services, which could be usually reused by different distributed systems.

Various communication middleware frameworks have been designed and developed, based on the available technological solutions. The following paragraphs introduce the most largely known and deployed. We briefly present their main characteristics and how they deal with integrability, interoperability, QoS and scalability requirements of distributed systems to justify the reasons why the Service-Oriented Architecture based on an Enterprise Service Bus is identified as the most adequate paradigm to manage requirements of distributed systems.

*Remote Procedure Call*

Remote Procedure Call (RPC) is a middleware based on inter-process communications allowing an application to execute a procedure or operation offered by a remote system [BIR 84] [RFC 1057], [RFC 5531]. It is mainly intended to deal with the integrability and was proposed to reduce the complexity of using directly transport and network services and to allow an entity to invoke a remote procedure, as it was local. The most popular RPC implementation is the Network File System protocol (NFS) proposed by Sun in the mid-80s [RFC 1094] in order to provide a transparent distributed file system.

RPC promotes limited interoperability support. It does not integrate adaptation and mediation solutions and the entities interact using specific APIs and languages implementations. Interface Definition Language (IDL) is used to allow communication between heterogeneous client and server [PIN 04]. However, RPC is closer to the Client-Server design model with synchronous request/reply interactions. This limits the large distribution, the offered QoS (only based on the used transport and network protocols) and scalability since a server centralizes the business logic [TAL 00].

*Object-Oriented Middleware*

The Object Oriented (OO) approach extends RPC and is mainly intended to deal with the integrability. It is based on Object Request Brokers (ORB) introduced to hide the complex use of transport and network services by allowing an entity to interact with an object as if they were in the same memory space [COM 13] [MIS 13] [RMI 13] [COR 13].

Well-known Object-Oriented implementations include: Distributed Component Object Model or COM/DCOM and .NET Remoting (Microsoft platforms) [COM 13] [MIS 13], Remote Method Invocation or RMI (Java platforms) [RMI 13], and the Common Object Request Broker Architecture or CORBA (multiple platforms and programming languages) [COR 13].

Even if it exists ORB solutions such as CORBA that offers interoperability functionalities to allow distributed objects written in multiple languages to interact, the OO approach promotes limited interoperability support and does not integrate adaptation and mediation solutions. The ORB solutions follow the Client-Server design model and support synchronous communications. Basically, they lack scalability and do not address QoS requirements [TAL 00]. Advanced ORB solutions supporting both synchronous and asynchronous communications, (e.g. CORBA v3) and dealing with QoS (e.g. Fault-Tolerant CORBA to deal with reliability) was proposed [EMM 00].

*Message-Oriented Middleware*

The Message-Oriented Middleware (MOM) is an approach based on message and channel concepts, offering asynchronous communications between distributed systems. The channel can be a queue for a point-to-point communication (one to one messaging style) or a topic to

support the publish/subscribe pattern (one to many, many to many or many to one messaging styles) (Figure 2.1).

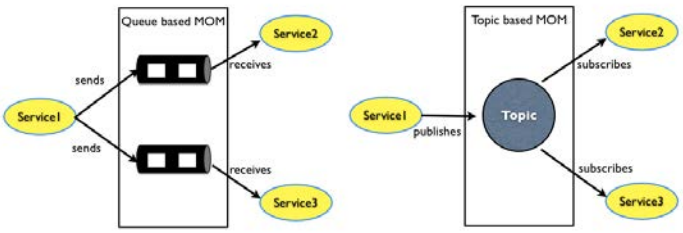


**Figure 2. 1 MOM-based integration**

A MOM is a good solution to satisfy integrability requirements and to develop distributed systems. However it does not satisfy interoperability requirements, since the consumers and producers need to be adapted to use the MOM solution messages formats and interfaces (e.g. Java Messaging Service - JMS). Also, the MOM solutions do not integrate adaptation and mediation mechanism. Advanced protocols, as AMQP (Advanced Message Queuing Protocol) are developed nowadays to deal with these interoperability issues [AMQ 12].

MOM also allows managing several QoS and scalability requirements. The Data Distribution Service for Real-Time Systems (DDS) [OMG 04], RabbitMQ [ALV 12] are examples of MOM-based implementations that allow enforcing QoS and scalability on distributed systems communications. For instance, the messages availability or the messages transfer reliability is managed with “data persistency” mechanisms. Interacting entities do not have to always be available in order to communicate since the MOM can temporarily store the messages to be processed later when the corresponding entities are available.

*Event-Driven Architecture*

The Event-Driven Architecture (EDA) is an architectural approach to implement a middleware that gives to distributed systems the ability to communicate in an asynchronous way through events exchanges [MIC 06]. EDA is an architectural paradigm that allows asynchronous communication and interaction between real-time applications. It gives a way to design applications or systems in which events flow between applications components, which can consume, detect and react to changes occurring in other applications components or in their contexts.

EDA is similar to the message orientation, but it is more interesting for systems that require a high reactivity or proactivity to deal immediately and quickly with a problem, an opportunity, etc.

Based on message oriented middleware technologies, EDA satisfies the integrability requirements, but not the interoperability requirements (see previous MOM paragraph). Astrolabe [BIR 02], Gryphon [PIE 03], Dream [BUC 04] are examples of EDA-based implementations that take into account the QoS and scalability requirements of distributed systems.

*Resource-Oriented Architecture*

Resource-Oriented model is an architectural approach to implement a middleware based on Representational State Transfer (REST) principles and using the protocol HyperText Transfer Protocol (HTTP) [FIE 00]. REST is an architectural style for integrating distributed entities by exposing their functionalities and data as web resources. Resources can be anything that can be available and accessible through the Web (files, pictures, business information, features, etc.). These resources are in general identified and indexed using Web URIs (Uniform Resource Identifiers).

REST can be seen as the most simple and lightweight middleware approach if the architecture of the targeted system is not complex. The integrability and interoperability support offered by REST are mainly based on the synchronous HTTP protocol with the well-known and basic GET, PUT, UPDATE and DELETE basic operations. However, because it uses the generic HTTP interface, REST does not allow having interfaces specific to complex applications (i.e. which would expose other operations that the GET, PUT, UPDATE and DELETE ones). In addition, without the web (HTTP protocol), REST becomes irrelevant since it does not support exchanges based on other synchronous or asynchronous communication protocols such as JMS, SMTP, FTP, etc. [FOU 08].

To finish, let us note that REST allows building Client-Server applications that present limitations in terms of large distribution, QoS and scalability [TAL 00].

*Service-Oriented Architecture*

The Service Oriented Architecture (SOA) [SOA 06] is an architectural approach for supporting integrability between distributed entities that share features and data as services. SOA guarantees also the interoperability as it promotes complete distributed solutions based on standard communication protocols and mechanisms to integrate various and heterogeneous platforms. Standard protocols such as HTTP are used for the communication together with other communication protocols (JMS, SMTP, FTP, etc.).

SOA offers an approach to build systems as a set of independent heterogeneous and distributed services available somewhere in the network and able to exchange data. These services can have potentially heterogeneous implementations (e.g. PHP, JavaScript, java, C/C++, BPEL, etc.) and can be composed in order to provide more sophisticated features and functionalities. The Web Service Description Language (WSDL) [WSD 01] [WSD 07] is used to describe the interface of functionalities exposed by services. Moreover, the Universal Description, Discovery and Integration (UDDI) [UDD 04] gives a standard way to publish and discover services.

Nowadays, the agile and efficient design and development of existing distributed systems mainly follow the Service Oriented Architecture paradigm with several technological frameworks and solutions proposed to provide required integrability and interoperability management functionalities between distributed and heterogeneous service providers and service consumers.

Next section details the different frameworks and solutions to manage well integrability and interoperability in SOA contexts.

### 2.2.3 Integrability and interoperability management in SOA contexts

The service-oriented approach and its architectural paradigm represent the most well-adapted solution to build distributed systems that will be easily extended by adding new business services or by integrating services provided by other systems in an interoperable and agile way.

However, the way the integrability and interoperability is managed within SoA-based IT infrastructures is not always the suited one. Indeed, one crucial challenge for IT infrastructure architects is to avoid ad hoc (or *accidental*) integration approaches.

Figure 2.2 illustrates the complexity involved in satisfying integrability and interoperability requirements. Let us assume here six heterogeneous services: three consumers and three providers (part I of figure 2.2). If we have to integrate consumers and providers (part II), each consumer needs to implement specific logic to adapt its data to the provider' formats and also to manage the interoperability. So, with three consumers and three providers, nine connections should be managed and nine "adapters" should be implemented to provide interoperability.

To generalize:

*Number of connections (integrability) and adapters (interoperability) = N (consumers) * M (providers)*

On figure 2.2 –II, the number is 3*3=9.

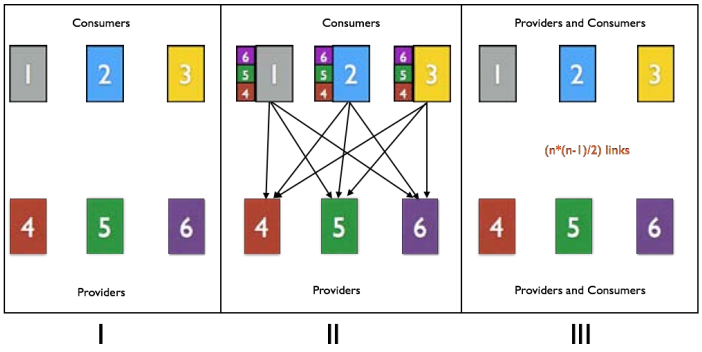


**Figure 2. 2 Example of integrability and interoperability complexity**

On the part III of figure 2.2, we assume now that the context is completely heterogeneous, and that each service is both consumer and provider of the others. It is the worse case where full integration and full interoperability are needed; in this case, fifteen "adapters" should be implemented and fifteen connections should be managed. More generally:

*Number of connections (integrability) and adapters (interoperability) = (N *(N-1)) /2 with N that represents the number of entities (consumers and providers)*

On figure 2.2 –III, the number is (6*5)/2=15

In order to avoid this kind of ad-hoc integrability and interoperability management approaches, several technical solutions have been proposed. These approaches are aimed at

reducing integrability and interoperability complexity by proposing mediation solutions based on Application Servers, Message Oriented Middleware, Enterprise Application Integration or Enterprise Service Bus.

*Application Server (AS)*

An Application Server can be considered as a middleware mediator as it provides services containers performing as runtime environments for hosting and running software components. These service containers are able to manage all the interactions between the hosted components but also the components lifecycle, resource allocation, security, etc. Most common application servers implement J2EE or .NET architectural frameworks (i.e. glassfish, tomcat, jboss, weblogic, websphere, IIS, etc.).

By using an Application Server, the complexity of managed connections between providers and consumers can be reduced because several provided services can be hosted in the same server. Specific URLs are used to identify provided services. Consumers just need to connect to the server and deal with the corresponding service URL (Figure 2.3). However they have to implement the adaptation logic to manage the interoperability.

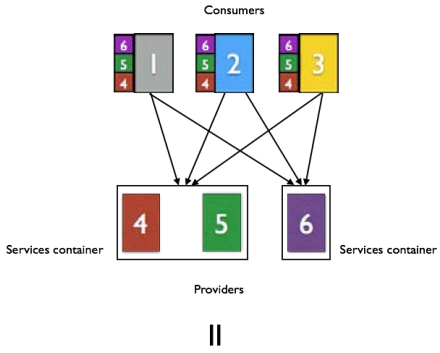


**Figure 2. 3 Application Server based integration**

Therefore, an Application Server reduces the complexity of integrability management. Interoperability still needs to be guaranteed by the implementation of specific adapters. Being a centralized solution, an Application Server has QoS and scalability issues.

*Message Oriented Middleware (MOM)*

As previously introduced, a MOM is a middleware based on message and channel message concepts and offering asynchronous communication between message producers and consumers. The channel can be a queue for a point-to-point communication (one to one

messaging style) or a topic to support the publish/subscribe pattern (one to many, many to many or many to one messaging style). A MOM is a good solution to satisfy integration requirements. However it does not satisfy interoperability requirements, as the consumers and producers need to be adapted to use the MOM solution messaging format and protocols. A MOM offers mechanisms to deal with the QoS (e.g. data persistency, error recovery, prioritization of message, etc.). However, it can have scalability issues when several message channels and transactions need to be managed [PIN 04].

*Enterprise Application Integration (EAI)*

Enterprise Application Integration server can be used as a middleware mediator by implementing the required adapters between consumers and providers based on the enterprise integration patterns [HOH 04].

Based on the XML language and the message brokers, EAI solutions manage the integrability required by distributed systems and include a set of adapters to guarantee the interoperability.

The figure 2.4-I shows how EAI reduces the number of connections to manage, as each actor has to connect to the central broker. However, interoperability is complex to guarantee. The central broker maintains duplicated adapters for each couple of heterogeneous consumer and provider. And too much effort is needed to develop and manage these specific adapters. For instance, based on the figure 2.4-II describing a scenario including three consumers and three providers, adding a new consumer implies adding three more adapters to allow this consumer to interact with the three providers if a full interoperability is needed.

EAI solutions offer a common interface based on proprietary technologies. This could increase interoperability complexity between products from different EAI vendors. And existing consumers and providers would require new adapters when a new EAI is installed. Moreover, EAI solutions are usually implemented based on a server acting as a central hub. And the main drawback of the EAI solutions is the high dependency on this central hub (i.e. potential scalability problems resulting from bottlenecks or risks of global system fault).

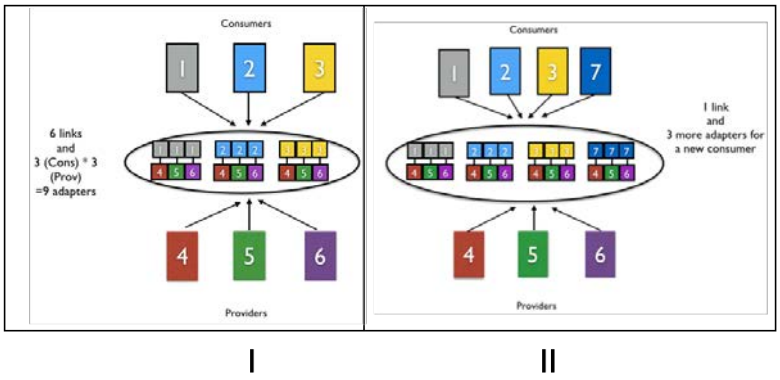


**Figure 2. 4 EAI-based integration**

*Enterprise Service Bus (ESB)*

Enterprise Service Bus solutions have been proposed as the best technical middleware solution that enables integration among distributed and heterogeneous components within SOA.

David Chappell defines them as *"standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across extended enterprises with transactional integrity." "An ESB applies web services and other complementary standards by combining them with technology concepts and best practices learned from EAI brokers"* [CHA 04].

ESB solutions represent an evolution of its EAI ancestor. By using standard protocols, ESB solutions ensure interoperability between heterogeneous systems. Based on a common normalized messaging approach, ESB solutions cope with the interoperability limitation of EAI solutions and facilitate the interoperability between products from different ESB vendors.

Moreover, ESB solutions offer features such as service discovery, intelligent routing, messages processing, service orchestration, etc. These capabilities are provided as services implemented by components distributed along the bus. These components can be centralized on a single computer or distributed across multiple interconnected computers. ESB solutions increase the QoS and the scalability by given distributed integration and mediation strategies.

The Figure 2.5 shows how ESB reduces the complexity of EAI. ESB offer standard interfaces to support several communication protocols. So existing consumers and providers do not require being adapted to use the ESB. Adapters implementing the supported interfaces are

maintained by the ESB and can be shared by the consumers and providers. For instance, based on the figure 2.5-II describing a scenario including three consumers and three providers, six adapters are used to manage the integrability. Consumers and providers use them to connect to the bus and to manage communication protocols interoperability. And adding a new consumer that uses a different protocol implies just adding a new adapter that supports the communication protocol of this consumer. In contrast to EAI, these adapters to deal with interoperability are not specific to each couple of service provider and service consumer since they can be shared.

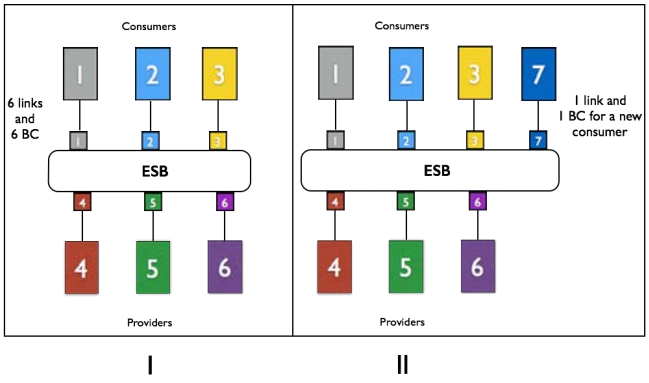


**Figure 2. 5 ESB-based integration**

Therefore, ESB solutions allow managing the integrability and interoperability of heterogeneous and distributed systems. And compared to centralized EAI solutions, ESB solutions improve the QoS and scalability since their components can be centralized on a single computer or distributed across multiple interconnected computers.

### 2.2.4 Summary

Several proposals and improvements were elaborated at network, transport and middleware layers to deal with distributed systems requirements.

Mechanisms and protocols were proposed to enhance the network and transport layers. Communication middleware solutions were proposed to deal more with the distribution and the heterogeneity of distributed systems.

The main conclusions taken from the study of evolutions and proposals at network and transport OSI layers are:

- proposals have been made to take into account only a subset of distributed systems requirements (QoS offered to distributed systems);
- proposals are not broadly deployed.

However, they inspire our approach to manage QoS and scalability requirements of distributed systems at the middleware layer.

The main conclusions taken from the study of the different middleware approaches are:

- the design following the Service Oriented Architecture paradigm can be considered as the one able to better satisfy the integrability and interoperability requirements of distributed systems;
- within the SOA, the Enterprise Service Bus is identified as the most efficient communication technology to integrate any kind of heterogeneous and distributed services and processes.

However, ESB solutions can present QoS and scalability issues when they have to support a high number of transactions. Indeed, an ESB is composed of a set of containers that host engines implementing the different integration and mediation functionalities. For these reasons, the ESB could represent a bottleneck and get congested or saturated, when a high number of transactions need to be exchanged. Congestion could appear due to the number of requests to be processed, but also due to the fact that the ESB runs on IT infrastructures with limited resources (CPU/threads, memory, network resources, etc.)

Next section 2.3 provides a deeper analysis of related works more specific to our thesis proposal. We present how these works enhance QoS, scalability, but also manageability and self-manageability of ESB solutions.

## 2.3 ESB-based communication middleware enhancements

Several approaches can be followed to improve proposals made at the middleware layers and more specifically the ESB solutions. In this section, works to extend ESB solutions to improve 1) QoS and scalability, and 2) manageability and self-manageability are presented.

### 2.3.1 ESB extensions for QoS and scalability management

Several proposals have been done to deal with the scalability and QoS of ESB solutions.

In general, the QoS offered by ESB is managed by improving the message routing process that is an important task of an ESB. It consists in the delivery of messages between a consumer endpoint and a provider endpoint according to a certain rule and logic. The routing rules can be statically fixed in a configuration file, driven by the content of the message, or can be guided by more smart strategies.

In [ZHO 10], the authors' proposal is a *Policy-Configurable Dynamic Routing* mechanism that allows users configuring dynamically routing policies. The authors' approach is based on a QoS model used by service consumers to express their requirements. This information is included in consumers' messages and is used by the ESB router to define the routing path dynamically at runtime. This proposal gives an interesting mechanism to improve the routing process. But the routing process only considers the consumers' requirements and not the providers' states.

In [WU 10], an extended ESB supporting dynamic routing based on the QoS level of the providers is proposed. The ESB has a mechanism to test the QoS level of the providers and to change dynamically the routing table. The mechanism allows improving the QoS in terms of reliability and availability, since the runtime states of the providers guide the definition of the routing rules. The proposed extension is driven by the provider contexts and can be improved by taking also into account the consumers' requirements.

In [WU 09], the authors define a redundant routing model that tolerates unavailability or failure of the services that have to deal with the consumer's requests. Indeed, traditional ESB solutions have a static routing service configuration and if the request is routed to a service that fails in its operation, the whole business function will fail. The proposed algorithm is to send the consumer's requests to two same services (a service A and its replica). The service and its replica are invoked at the same time and their responses are aggregated and compared to detect error. The good response is sent to the consumer. This proposal increases the number of successful responses at the consumer's point of view but has some shortcomings:

- it is mandatory to have services and replicas, and this is not always applied for all the services;
- there is no guarantee that the service and its replica will not fail at the same time;

- when the service and its replica exist, the router needs to have the response of both before sending the good one to the consumer. The impact on the response time can be high if the service and its replica don't have the same performance;
- the ESB routing process is complex and can become the point of failure when the traffic through the ESB is high.

Extensions and mechanisms have also been proposed to give to the ESB the ability to guarantee the QoS by controlling the load through the bus or the traffic sent to the service provider.

Authors of [LIU 08] have proposed an architecture aimed at executing and coordinating multiple mechanisms for managing the QoS offered to distributed systems. Two interesting mechanisms are introduced. The first is a failover mechanism to reroute traffic to an alternative provider if the initial provider presents issues (such as a high response time). The second mechanism allows controlling the provider overload by applying throttling strategies to regulate the requests flow to be processed before invoking the service providers. This mechanism increases the reliability by avoiding request losses at the provider side. However:

- the proposed mechanisms do not prevent losses within the ESB itself. It is then needed to add mechanisms that deal with issues taking place at the ESB side (overload of the ESB components and communication channels);
- also, the authors do not precise how to choose the mechanisms to use according to the issue; while efficient approaches and strategies are needed to coordinate the mechanisms according to the occurring issues need to be added.

Regarding the scalability requirement, it is more guaranteed by overprovisioning, redundancy and load balancing techniques with several clustered or federated instances.

Authors of [CAL 08] propose protocols (IFP – Interior Federation Protocol and EFP – External Federation Protocol) to deal with the scalability by managing federations of ESB instances. The proposal is based on the possibility for an ESB instance to dynamically join a federation. The collaboration of the federated instances is made using distributed service registries. Each ESB has its own registry that contains the description of services it references. This information is shared to have a converged view of available services within the

federation. The followed approach to manage the federation is efficient in the actual context of networked environments. However:

- the number of messages that federation members have to periodically exchange to share registries information can create scalability issues;
- to this limitation, we can add the complexity to manage the consistency of distributed knowledge in dynamic contexts where a large number of concurrent services can be provided or removed.

In [SIL 11], authors motivate the introduction of a Manufacturing Service Bus (MSB). To overcome the disadvantages of a single ESB in manufacturing contexts, authors distinguish the different phases of products lifecycle and propose one ESB instance for each phase. The phase-specific ESB instances are connected by a hierarchic ESB called PLM-Bus. By this way, each ESB can be adapted individually to the requirements of the corresponding phase (availability, throughput, time-constraints, etc.). Even if this approach increases the ESB instances to manage, it allows a comprehensive load distribution and improves the scalability. However the coordination and collaboration of the multiple ESB instances need to be well managed.

The Pervasive Service Bus (PSB) proposed in [PAN 12] follows a similar approach, by using a distributed and federated topology to manage the scalability. Intelligent mechanisms are added to deal with underlying computing resources.

Next section presents proposals enhancing the manageability and the self-manageability of ESB solutions.

### 2.3.2 ESB extensions for manageability and self-manageability satisfaction

The management of QoS and scalability requirements can be done either in a static way or in a dynamic way. Hereafter, approaches and frameworks that were proposed towards a dynamic and self-manageable ESB able to be self-monitored and self-controlled in order to adjust itself in dynamic environments are presented.

In general, proposed solutions to have a dynamic ESB focus on the execution of adaptation mechanisms on the mediation flow defined as a set of mediation modules or components

through which messages transit during the integration and that provide mediation functions between service consumers and providers (router, adaptor, etc.).

In [GON 11] and [GON 13], authors propose an adaptive ESB infrastructure in order to address QoS issues and to support self-adaptive service-oriented systems with a framework that gives dynamic adaptation capabilities to an ESB. Authors integrate in their solution a monitoring and execution framework presented in [KAZ 09]. The main goal of this framework is to detect events (mainly the response time degradation or saturation of service providers) and to execute adaptations (mainly the modification of the routing process and the switching of the request to an equivalent service provider). However, the proposed approach presents some limits since all the messages are intercepted to check if the routing process need to be adapted or not. This approach does not scale well in the actual ESB deployment contexts with a high number of transactions since the interruption of the message includes latency on the communication. Also, proposed monitoring and execution mechanisms focus respectively on the service provider's state and the routing process while they can be applied on different levels (e.g. ESB component parameters, services providers, running business processes, exchanged messages, underlying computing resources, etc.) to satisfy QoS and scalability requirements of distributed systems. Finally, the proposed framework does not integrate an analysis phase of the monitoring metrics and a decision phase to well define the adaptation plan.

In [MAS 10], the authors propose an architectural specification and some functional models and mechanisms for an adaptive ESB. The proposal is based on three main components: a monitoring component that determines the state of the ESB, a management component to guide the execution of services deployed on the ESB, and a policy engine to define the service adaptation. The proposal is similar to the one done in [GON 11]. Proposed adaptation mechanisms focus only on the modification of the messages route, but authors of [MAS 10] integrate a policy engine to guide the adaptation plan.

In [MOR 13], extensions to the integration framework called Cilia [CIL 12] are proposed. The main goal of proposed extensions is to make dynamic the creation and management of integration process chains based on execution contexts evolutions. The theory of control is used with several state variables, which allow following the state of interconnected components, and action variables, which guide the dynamic adaptation of the mediation flow for instance when a new component is added.

### 2.3.3 Summary

Several solutions for the management of the QoS and scalability of ESB were proposed. In general, scalability of ESB is managed using redundancy, overprovisioning and load balancing techniques using several instances. Extensions and mechanisms were also proposed to give to the ESB the ability to guarantee the QoS by controlling the load or by improving the routing process.

However, the proposals present limits since they are exclusively intended to the management of the QoS or the scalability; and not to both at the same time. Also, the proposals for the QoS management only deal with part of the identified QoS parameters (e.g. only the reliability or only the latency) and can create other scalability and QoS issues (e.g. the replica solution proposed in [WU 09]).

However, existing proposals are mainly based on an extended ESB with static hard-wired mechanism configured without manageability and self-manageability capabilities required in the dynamic context of distributed systems.

Existing improvements to have a dynamic ESB mainly focus on the monitoring of the state of deployed services and execution mechanisms that adapt the mediation flow. However, to have a dynamic and self-manageable ESB solution, more efficient and scalable monitoring and execution mechanisms able to monitor and control ESB parameters, deployed services and business processes, exchanged messages and underlying computing resources are needed.

Also, it exists a lack of a mature architecture that well integrates and separates all the needed functionalities to have a self-manageable system namely [IBM 03]:

- a monitoring process to get information related to the state of the whole ESB and its underlying computing resources;
- an analysis process to determine a need for change on the managed system;
- a plan process to retrieve a set of strategies to apply for the needed change;
- an execution process that performs actions related to the defined plan, step by step.

## 2.4 Conclusion

Many communication requirements have appeared with the evolution of modern and distributed systems. In this chapter, a state of the art of existing solutions targeting these requirements has been presented.

In a first section (Section 2.2), we have presented the evolution of network, transport and middleware layers solutions. We have presented how these solutions deal with identified communication challenges and requirements.

In a second section (Section 2.3), we have introduced related works more specific to how existing ESB solutions have been improved.

The presented state of the art allows us justifying the introduction of our proposal, the Autonomic Service Bus (ASB), as a new generation of communication solution well suited for modern distributed systems.

Based on an open and standard ESB solution guaranteeing integrability and interoperability requirements, we propose a specialization targeting the management of the QoS, scalability, manageability and self-manageability requirements.

Hereafter we show briefly how the ASB will enhance the conceptual lacks of existing related works:

- in contrast to proposals for QoS and scalability management are based on one mechanism intended exclusively either to the QoS management or to the scalability management and not to the both at the same time

    - the ASB will be based on a set of mechanisms for QoS and scalability satisfaction. Virtualization and cloud computing mechanisms (self-provisioning, elastic resources, clustering, federation, etc.) will be exploited to avoid overprovisioning techniques and to have efficient solutions for scalability management. In addition to these solutions exploiting the virtualization and cloud technologies, an important number of mechanisms inspired from the transport and network layers will be proposed in order to take into account distributed systems QoS requirements.

- the proposed solutions in the literature for QoS management are statically configured based on mechanisms hard-wired to the ESB to only deal with specific part of the identified QoS parameters (e.g. only the reliability or only the latency). Such configurations fail in correctly addressing in a dynamic way the different QoS expectations of distributed systems. Moreover, in the dynamic contexts and environments where ESB solutions evolve and operate, managing the scalability and QoS requirements (with actions such as mechanisms selection, coordination, composition, configuration, reconfiguration, etc.) is a too complex task when it is done manually. To cover this lack of proposals for a self-manageable ESB:

    - the ASB will have the ability to manage itself its scalability and QoS-oriented properties. The standard framework proposed by IBM to implement the autonomic computing paradigm and build autonomic systems will be used. Advanced models based on semantic technologies and probabilistic approaches will be exploited to guide the self-properties of the ASB;
    - proposed mechanisms for QoS and scalability will be well defined and characterized. Moreover, flexible design solutions will be followed to allow the selection, coordination, composition, configuration, reconfiguration and use of most adequate mechanisms according to QoS and scalability properties to satisfy.

Next chapter 3 presents the design of the architectural framework of the ASB.

# Chapter 3. **Autonomic Service Bus Architectural Framework**

**Contents**



## 3.1 Introduction

The Service Oriented Architecture (SOA) based on an Enterprise Service Bus (ESB) is considered as the best paradigm to satisfy integrability and interoperability requirements of distributed systems by dealing with the distribution and heterogeneity of services, and applications. However, current ESB implementations are not able to guarantee in a smart and autonomic way QoS and scalability requirements of large distributed systems deployed in high dynamic environments. In this context, the Autonomic Service Bus (ASB) is proposed as a new generation of ESB solutions able to manage in an autonomic way the integrability and interoperability required by the current distributed systems by taking into account the QoS and scalability requirements.

The main goal of this chapter is to detail the proposed architectural framework of the ASB. Proposed mechanisms for QoS and scalability management are introduced and the architectural design of the ASB is detailed.

Several architectural design requirements come from the proposal namely:

- a flexible and extensible architecture to allow the easy introduction of mechanisms for QoS and scalability requirements.
- a mature architecture to allow satisfying manageability and self-manageability;
- a generic architecture to be implemented using any existing ESB implementations;

For the flexibility and extensibility of the proposed architecture, a JBI compliant ESB implementation deployed on a “cloud-oriented” infrastructure (a flexible virtual infrastructure based on a physical server supporting a hypervisor in order to manage virtual machines and java virtual machines to host the ESB) is considered. The “cloud-oriented” infrastructure gives us the possibility to manage in a flexible way the underlying computing resources used to host the ESB. The Java Business Integrator (JBI) specification proposed by the Java Community Process (JCP) in the JSR 208 specification [TEN 05] to define a platform for building ESBs using a set of plug and play services components is followed to have an open and extensible implementation. JBI allows integrating easily proposed QoS mechanisms as a set of pluggable components.

To tackle manageability and self-manageability requirements, the Autonomic Computing framework introduced by IBM [IBM 05] is followed. This framework allows having a self-manageable solution with the ability to adjust itself, to be self-monitored and self-controlled.

To design this self-manageable solution in order to be implemented using any kinds of ESB implementations, the Model Driven Architecture (MDA) [SOL 00] specification and design methodology is followed. It allows having a generic, portable and understandable architecture that can be instantiated using any kinds of ESB implementations.

The chapter is structured as follows: section 3.2 introduces the architectural foundations of the ASB. Section 3.3 introduces the proposed mechanisms for QoS and scalability. Section 3.4 presents the architectural overview of the ASB and its UML models. Section 3.5 introduces the design of the platform used for the ASB implementation. Finally, section 3.6 gives the conclusions of this chapter.

## 3.2 ASB architectural foundations

The main architectural foundations of the ASB design detailed in the following sections are the autonomic computing framework, the MDA design methodology, the JBI specification and the cloud computing paradigm.

### 3.2.1 Autonomic computing

The proposed ASB needs extensions and self-management capabilities to guarantee QoS and scalability requirements of large and complex distributed systems. Several approaches can be followed to design and develop such kinds of dynamic systems able to change their behaviour based on high-level policies. For instance, approaches based on agents-oriented architectural models were proposed with entities capable to perform actions in order to meet their design objectives [WOO 95] [TRI 07] [BRA 09]. IBM has introduced the autonomic computing framework [KEP 03] based on an autonomic manager, which implements a control loop based on four functions (monitoring, analysis, plan and execution) guided by predefined policies.

In this thesis, this standard and well-known autonomic computing framework proposed by IBM is followed for the ASB design. It offers a framework to enhance components that are not autonomic and to integrate the required components, processes, interfaces and decisions models to make them autonomic. The IBM framework also provides an incremental process to make a system autonomic. The following paragraphs give more details about this framework.

The Autonomic Computing framework is a self-managing computing model proposed by IBM to enable systems to manage and maintain themselves autonomously like the human immune system. To implement this autonomous behaviour, the framework has introduced the “Autonomic Manager” concept, as an entity able to manage a component or a set of system components named “Managed Elements”. This management is done on the basis of a set of policies (Knowledge Base – KB) predefined at design time or learned at runtime. “Autonomic Manager” (AM) and “Managed Elements” (ME) are associated to define “Autonomic Elements” (Figure 3.1).

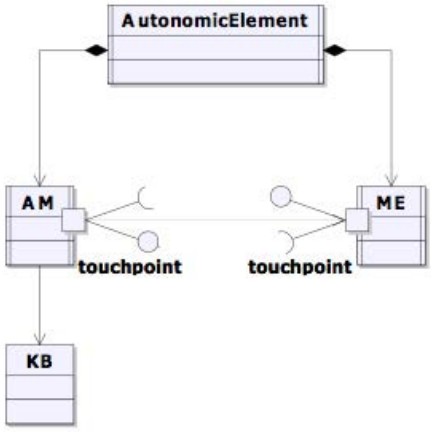


**Figure 3. 1 Autonomic Element Architecture**

The instantiation of Autonomic Computing paradigm is based on the implementation of a control loop named "MAPE" (Monitoring, Analysis, Planning and Execution). This control loop implies:

- a Monitoring process allowing getting information related to the state of the whole system at different levels. This information needs to be aggregated, filtered and correlated to detect and identify a symptom (e.g. performance and scalability issues);
- an Analysis process allowing determining a need for change on the managed system. A set of reasoning or prediction approaches, following for instance machine-learning techniques or probabilistic models, can be applied to identify the cause of the detected symptom. A good diagnostic from the analysis process will guide the needed actions to perform;
- a Planning process allowing retrieving a set of strategies to apply for the needed change. For instance, the application of a plan will allow dealing with a QoS degradation;
- an Execution process that performs step by step actions related to the defined plan.

Furthermore, the Autonomic Computing paradigm implies also:

- the implementation of "touchpoint" interfaces allowing the communication between the Autonomic Manager and the Managed Elements. "Sensors" are used by the Autonomic Manager to get the state of the managed elements and "actuators" to indicate and/or apply the adaptation actions to managed elements;
- the implementation of a knowledge base. This base can include actions to be implemented to manage a component or a system. Likewise, all the MAPE loop entities use this knowledge base to perform their specific functions. The management at runtime of the

system is done according to a set of policies defined as part of the shared knowledge base that contains data providing syntax and semantic description of the systems, but also the rules to be applied in order to trigger actions that implement the self-management functions and the autonomic behaviour.

The implementation of the MAPE control loop to make a basic system autonomic is not a simple task. Five levels of autonomic maturity are proposed IBM to identify the transition steps for an incremental implementation from a basic and manual system to an autonomic solution (Figure 3.2) [IBM 05].

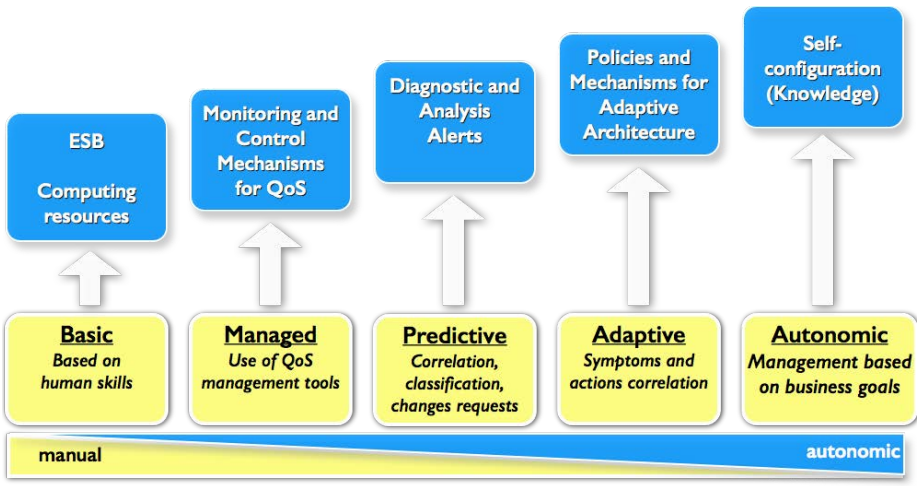


**Figure 3. 2 Autonomic maturity levels**

*Level 1: Basic*

At the basic level, the configuration and management of the system are done separately and require extensive human skills. The administrator has to monitor the system, to analyse the observed metrics and eventually to act on the system following a defined plan. All these actions are done manually using default solutions such as an administration command line or a web based management console.

*Level 2: Managed*

At the managed level, specific monitoring tools are used. Metrics are collected, aggregated, correlated and filtered to detect anomalies (symptoms). A set of scripts and mechanisms are used to act less manually on the system to correct symptoms. However, the administrator has to analyse the detected symptoms and to initiate the specific corrective actions to execute.

*Level 3: Predictive*

The predictive level includes diagnostic and analysis alerts solutions. These solutions classify symptoms detected by the monitoring and recommend changes to apply. The administrator has to approve (or not) recommendations and to initiate corrective actions.

*Level 4: Adaptive*

At the adaptive level, the administrator does not have to approve (or not) recommendations and to initiate corrective actions since it defines policies and mechanisms that correlate symptoms and actions. In other words, the system is able at this level to take actions alone based on rules defined by the administrator. Defined rules at the adaptive level are more oriented to the IT concepts and parameters.

*Level 5: Autonomic*

At the autonomic level, the system is able to take alone actions based on business level requirements defined by the users (and not only IT level requirements). The knowledge to traduce the business level requirements to IT level policies needs to be included in the system by the administrator. At the end of this level 5, the basic system becomes autonomic.

In this thesis, these five levels of maturity are followed to identify the transition steps for an incremental design and implementation from a basic and manual ESB to an autonomic ESB. We have firstly developed monitoring and execution solutions with the needed “touchpoint” interfaces (sensors and effectors). And secondly, solutions are proposed to cover analysis and plan actions.

Next section presents MDA specification followed to have a generic architectural design.

### 3.2.2 Model-driven methodology for generic software design

In order to have a generic, portable and understandable architecture that can be instantiated using any kinds of ESB implementations, the Model Driven Architecture is the followed methodology to design the ASB.

*Model Driven Architecture*

The Model Driven Architecture approach [SOL 00] is used to design a generic architecture that can be instantiated using different technical solutions. This approach is based on a set of

specifications intended to make architectures portable, interoperable and reusable. These specifications, which should be applied during design and development phases, involve the use of multiple models to represent and describe: the specifications and logic of the system, the system operations, and finally the technical concepts required to deploy the system on a specific platform.

Therefore, we can distinguish three levels of models in this approach namely:

- the Computational Independent Model (CIM), which is used to describe how the system behaves in its environment by representing independently of any computer system the expected functionalities and the different use cases;
- the Platform Independent Model (PIM), which is a conceptual computer-oriented representation of the system specifications (generally based on UML). This model describes the structure and operations of the system regardless of the execution platform;
- the Platform Specific Model (PSM), which describes the structure and operations of the system taking into account the technical concepts and constraints that are imposed by the chosen execution platform.

Models at the different levels are proposed to have a generic and portable architecture of the ASB.

Next section introduces the Java Business Integration (JBI) specification and the cloud computing adopted for the flexibility and extensibility of the proposed solution.

### 3.2.3 Solutions for a flexible and extensible architecture

*Java Business Integration*

One main open standard has been specified to provide architectural guidelines for designing and developing ESB solutions namely the Java Business Integration (JBI) specification. The JBI specification has been proposed by the Java Community Process (JCP) in the JSR 208 specification in order to facilitate the development of a common and compatible fundamental architecture for ESB solutions implementation [TEN 05].

JBI is defined as a set of plug and play web services components (providers and consumers of services) that communicate via a Normalized Message Router (NMR). External components

can become providers or consumers connected to the JBI container via the Binding Components (BC). Internal components can be hosted within the JBI container via the Service Engines (SE) in order to provide mediation services (Figure 3.3).

JSR 208 specification defines the Binding Components as entities that are able to provide transport protocol independency to allow communication between JBI components and other protocol-dependent components. For instance, within an ESB you can implement a binding component able to connect to an external FTP repository service to send or retrieve files and offering a Web Service interface (WSDL). Such FTP binding component will be used by other JBI components in order to independently communicate with the external FTP service. Examples of binding component implementations are FTP, SFTP, Email or SMTP, JDBC, JMS, File, HTTP/SOAP, REST, SMPP (via SMS), XMPP (instant messaging), RSS (feeds), SIP, etc.

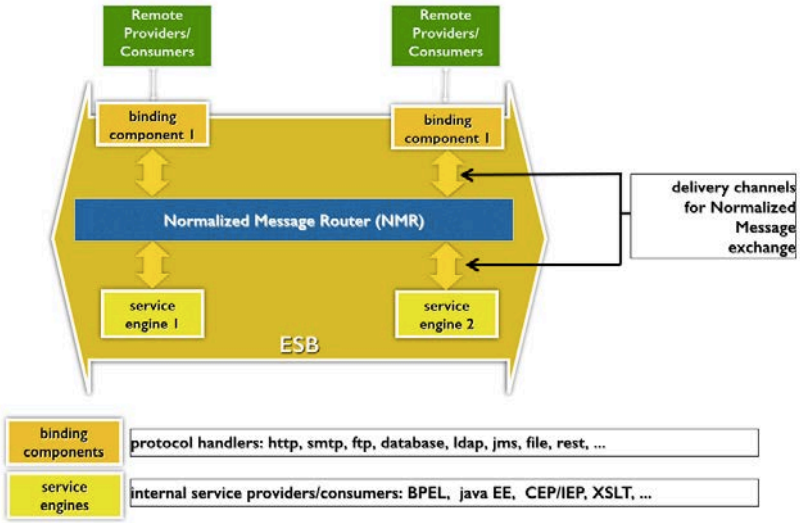


**Figure 3. 3 Java Business Integration (JBI) Architecture**

JSR 208 specification defines the Service Engines as containers allowing to instantiate service units implementing specific business logic. Service units are intended to collaborate with other JBI components by using a web service interface (WSDL). Examples of service engines are BPEL, Java EE, XSLT, SQL and data mashup, IEP, ETL, etc.

Binding components and service engines interact through a Normalized Message Router (NMR) responsible for transmitting Message Exchanges; and the communication between these components and the NMR is performing using delivery channels.

In this thesis, our implemented solution is based on a JBI-compliant ESB implementation. JBI satisfies needed flexibility and extensibility design requirements by allowing the integration, configuration and reconfiguration of proposed mechanisms for QoS and scalability management as a set of pluggable components.

*Cloud Computing*

As its main predecessors or related technologies (mainframe, workstation, Grid Computing, Utility Computing, etc.), cloud computing is a paradigm, a technological style aimed at providing in a flexible and easier way hardware or software-computing resources as services. Cloud computing is defined by Gartner as “a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service using Internet technologies” [GAC 14].

The cloud computing offers a set of ready to use resources accessible either from a client web browser or command line console. According to the type of resources we have different categories of services. The fundamental service layers defined by the NIST are [NIS 11]:

- Software as a Service (SaaS) that consists in providing to cloud consumers applications and / or software to process business functions;
- Platform as a Service (PaaS) that consists in providing to cloud consumers platforms that allow them developing, deploying and hosting their own applications and software;
- Infrastructure as a Service (IaaS) that consists in providing to cloud consumers infrastructural resources (storage, computing, network) usable for deploying and running their own platforms, applications and software.

Cloud computing allows making a system highly flexible, available and reliable based on supported operations such as self-service and elasticity. It is based on mechanisms and strategies that allow guaranteeing the five essential characteristics [NIS 11]:

- on-demand self-service: A cloud consumer can provision, use and release resources based on its needs and these operations don’t require intervention at the provider side. This offers more agility to cloud user;
- broad network access: A cloud consumer can have access to providers’ resources and services using any kind of devices (smartphone, tablets, laptops, etc.) connected to the Internet;

- resource pooling: A cloud provider applies multi-tenant models to pool its resources to serve multiple cloud consumers. This operation allows providers using better their resources and need to be transparent for consumer;
- rapid elasticity: A consumer can have access to the services at any time, as they have an unlimited capacity. At the provider side, this requires an adaptation of the underlying resources and a dynamic allocation according to the load and the number of consumers;
- measured service: Cloud systems automatically control and optimize resource usage by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the used service.

In this thesis, a “cloud-oriented” infrastructure is considered as the deployment context of the ESB solution to give us the possibility to manage in a flexible way the used underlying computing resources.

#### 3.2.4 Summary

The main design foundations followed in this thesis to satisfy the architectural design requirements of the proposed ASB have been presented in this section with:

- the Model Driven Architecture (MDA) [SOL 00] specification and design methodology for a generic architecture to be implemented using any existing ESB implementations;
- the Autonomic Computing framework introduced by IBM [IBM 05] for a mature architecture that satisfies manageability and self-manageability;
- the Java Business Integrator (JBI) specification proposed by the Java Community Process and the cloud computing paradigm for a flexible and extensible architecture that allows the easy introduction of mechanisms for QoS and scalability requirements.

Next section introduces a catalogue of mechanisms proposed to extend ESB solutions and the underlying computing resources for QoS and scalability satisfaction.

### 3.3 Mechanisms for QoS and scalability management

In the literature, different mechanisms were proposed to extend ESB solutions in order to satisfy QoS and scalability requirements [CAL 08] [LIU 08] [WU 09] [ZHO 10] [WU 10] [SIL 11] [PAN 12]. The proposals, detailed in chapter 2, are mainly based on:

- extended ESB with hard-wired mechanisms that have static configurations. For instance, mechanisms were proposed to control the load of ESB components and communication channels or to improve the ESB router component behaviour;
- redundancy and load balancing mechanisms based on several ESB instances or duplicated hosted services and processes, etc.;
- overprovision techniques generally used to deal with the configuration and the state of the physical server that hosts the ESB instances.

However, proposals present limits since:

- static proposals are not suitable in the dynamic context of distributed systems;
- overprovisioning techniques do not deal efficiently with the requirements since allocated resources are not efficiently used;
- proposals are intended exclusively either to the QoS management or to the scalability management and not to both at the same time;
- proposals for the QoS management deal only with part of the identified QoS parameters (e.g. only the reliability or only the latency).

In contrast to these proposals, more efficient mechanisms that can be added to the ESB and its underlying computing resources are proposed in this thesis to deal with both QoS and scalability. They are classified in two categories according to the targeted layers:

- mechanisms that target the ESB implementation (the configuration, state and behaviour of the ESB components and communication channels but also of supported services and processes) are inspired from our background in transport and network-oriented solutions for QoS management (e.g. congestion control, error control, shaping, differentiation, resource reservation, admission control, etc.). For instance, when the ESB has to integrate a large number of parallel and concurrent systems, the congestion control mechanism can be deployed within the ESB to avoid congestion of communication channels and to limit the usage of a saturated service provider. These mechanisms will be classified as intra-bus mechanisms;
- mechanisms that target the underlying computing resources are based on virtualization and cloud-computing characteristics (e.g. clustering, federation, load balancing, elasticity, self-provisioning, live migration, etc.) to avoid the overprovisioning or oversized techniques

and to well manage resources of the IT infrastructures. For instance, the elasticity mechanism is applied on the underlying computing resources (e.g. processor, memory, storage resources, etc.) according to running transactions. These mechanisms will be classified as extra-bus mechanisms.

Next paragraphs present the design of intra-bus and extra-bus mechanisms proposed for the ASB to guarantee the global QoS and scalability.

### 3.3.1 Intra-bus mechanisms

To guarantee the QoS and scalability requirements, a lot of mechanisms can be added to the ESB implementations.

One of the original contributions of this thesis is the proposal to integrate at the middleware layer (ESB solution) mechanisms that are inspired on classical and enhanced QoS-oriented mechanisms currently used within the network and transport protocols. These mechanisms can be dynamically integrated to the basic behaviour of the ESB during the mediation processes to take into account requirements and constraints of distributed systems by implementing following functionalities:

- a flow control mechanism to limit the rate at which requests are sent through the ESB to saturated service providers. Thanks to this mechanism, it is possible for the ESB to test the providers' state before transferring consumers' requests and even before accepting them. Implementation of this mechanism can consist in delaying consumers' requests if the provider is saturated (Figure 3.4);

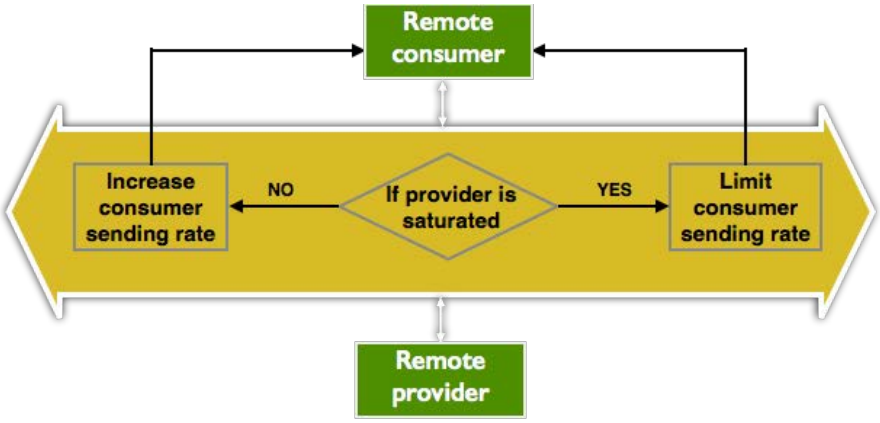


**Figure 3. 4 Flow control mechanism**

- a congestion control mechanism (almost similar to flow control mechanism) to limit the rate at which requests are processed through the ESB in order to avoid saturating the internal ESB components and communication channels;
- a differentiation and prioritization mechanism to manage the QoS offered to parallel and concurrent transactions with different priorities and QoS requirements. Thanks to this mechanism, it is possible for the ESB to differentiate either requests coming from several consumers or requests that are intended to several providers. To differentiate requests coming from several consumers, implementation of this mechanism can consist in giving a priority to each consumer and this priority will guide the way the mediation process is done by the ESB (Figure 3.5 - I). To differentiate requests that are intended to several providers, implementation of this mechanism can consist in giving a priority to each provider and this priority will guide the way the mediation process is done by the ESB (Figure 3.5 - II);

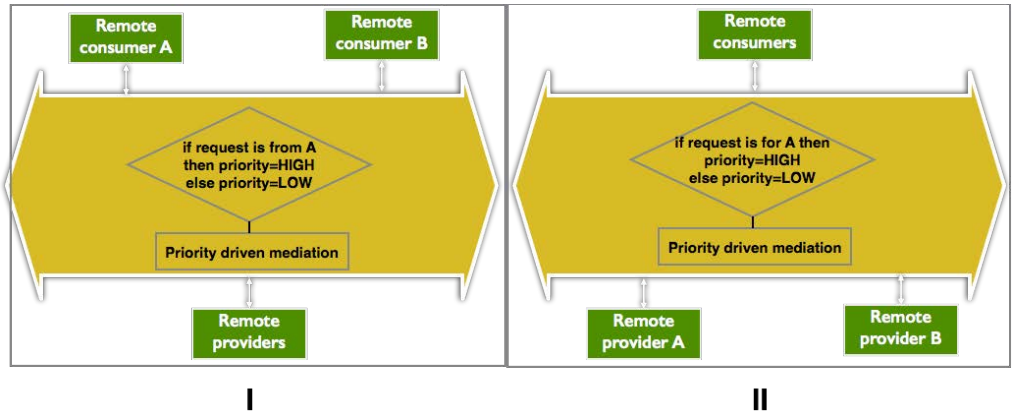


**Figure 3. 5 Prioritization and differentiation mechanism**

- an admission control and resources reservation mechanisms to guarantee a certain level of QoS. Thanks to the admission control mechanism, it is possible to get the current state of the ESB and to test whether the ESB is able or not to accept more incoming requests. The resources reservation mechanism can guide the differentiation and prioritization mechanism by allowing defining the behaviour of the ESB mediation components according to specific transactions;
- a partial reliability mechanism to respect the response delay constraints of critical requests. Thanks to this mechanism, requests of least importance can be eliminated at the entry point of the ESB when it starts getting saturated;

- a cache-based mechanism to keep in memory the responses of providers (that are the same for a defined period) and to deliver them to the future requests. Thanks to this mechanism, it is possible for the ESB to give a response to a consumer without invoking the provider. Therefore the latency is improved;
- an error control mechanism to perform error detection and to provide error recovery. An error can be detected when a service is unavailable or when a request or a response is lost. For instance, the traditional behaviour of the ESB is to route an error message to the consumer when a provider takes too much time to respond. Thanks to this mechanism, it is possible for the ESB to search for an equivalent provider. If such provider exists, it performs a retransmission of the request to this equivalent one. Only when an equivalent service provider does not exist, a notification will be sent to the consumer.

Let us recall that this list is not exhaustive, and that all of these mechanisms may be extended and specialized in order to satisfy specific combinations of QoS requirements. Hereafter we further detail how one among the proposed mechanisms (i.e. network inspired flow control mechanisms) is useful and how it can be designed and specialized.

**Flow Control.** The flow control mechanism is used to limit the sending rate over the ESB in order to avoid exceeding the buffering capacity of service providers and ESB communication channels.

The result of an experiment showing the impact of a provider capacity on the ESB QoS (i.e. the propagation time of a request through the ESB) is presented in the figure 3.6. Details of the used evaluation platform are presented in section 3.5.

On the figure, the propagation time corresponds to the time that requests spent through the ESB. The provider capacity is the number of requests that the provider can process in concurrence. The provider capacity has been set to values comprised between 5 requests (upper red line), 10 (middle green line), and 250 (lower blue line) requests. For each configuration, the consumer sends 40 requests.

Based on the analysis of the figure, we can say that the capacity of the provider has a real impact on the ESB propagation time. Indeed if the capacity of the provider is inferior to the number of sent requests, the propagation time through the ESB will increase considerably. This observation may be explained by the fact that when the provider is not able to receive

requests, the ESB will store them until the provider becomes able to process them. During this period, the ESB uses its resources (e.g. threads that have to accept the incoming requests) and may reach a state when it will not be able to accept more requests. In this context, a flow control mechanism is really needed to limit the rate at which requests are sent to saturated service providers.

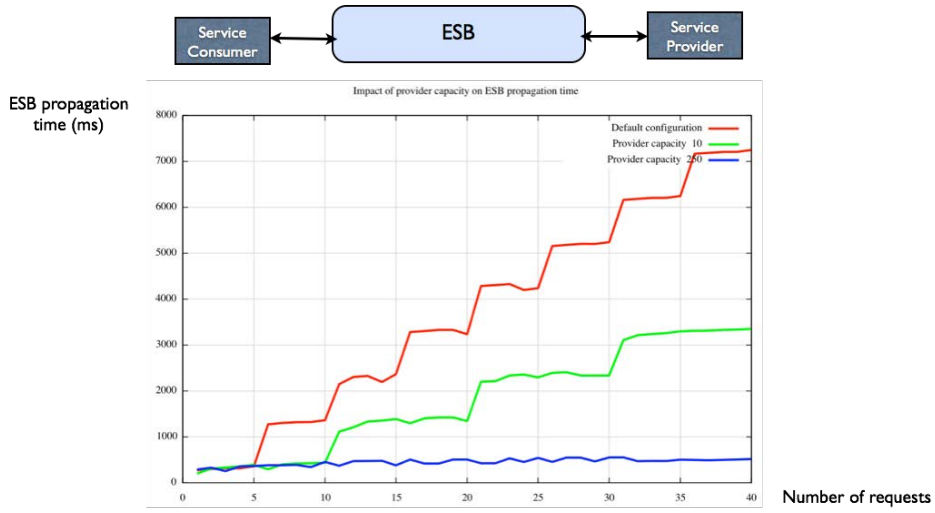


**Figure 3. 6 Impact of a provider's capacity on the ESB propagation time**

The flow control can be a smart micro-protocol based on the way it performs its tasks when provider saturation situations occur. For instance:

- the flow control can be basic by refusing all incoming requests. This approach is similar to the technical throttling solution that exists in several ESB implementations and it can be improved;
- a second approach to perform the flow control is to notify the service consumer about the provider state. The service consumer can then delay the sending message process. By this way, ESB actions are reduced;
- a third approach is to accept the requests but to put them into an external queue or message broker until the provider becomes ready to process them. The service consumer can be informed with an estimation of the time that its requests will take before being processed. It can decide if this time is acceptable or not;
- the fourth and last approach can be a "Priority driven selective limitation" or "Priority driven selective discarding" with the possibility for the ESB to accept or limit requests based on their priority. It can adapt the flow control by selectively discarding or queuing

requests of least importance. For this approach, service consumers have to send to the ESB the priority of their requests.

The next section introduces the extra-bus mechanisms mainly based on the self-provisioning and elasticity concepts of the virtualization and cloud computing characteristics.

### 3.3.2 Extra-bus mechanisms

Extra-bus mechanisms are based on virtualization and cloud computing characteristics (e.g. elasticity, self-provisioning, clustering, load balancing, federation, live migration, etc.) to avoid the overprovisioning or oversized techniques and to manage well resources of the IT infrastructures. So following the cloud computing, these characteristics are applied to satisfy the scalability in an efficient way:

- the vertical elasticity concept is applied to add or reduce resources allocated to the virtual machine or the java virtual machine on which runs the ESB;
- whereas the horizontal elasticity concept is applied when it is more interesting to deploy a new ESB instance and create a cluster (for a high availability and an efficient load balancing) or a federation (to increase the distribution of the mediation processes and components).

*Vertical elasticity*

We develop several mechanisms to apply the vertical elasticity concept on the computing resources (virtual machine, java virtual machine and ESB components resources):

- a mechanism to allow tuning (adding or removing) resources of virtual machines according to the resources usage level. Implementation of this mechanism can consist in dynamically allocate or remove CPU and RAM to the virtual machines that host the ESB according to the fluctuation of transactions through it (Figure 3.7). For instance allocated memory to the virtual machines is increased when we have a high memory usage due to the number of supported transactions;

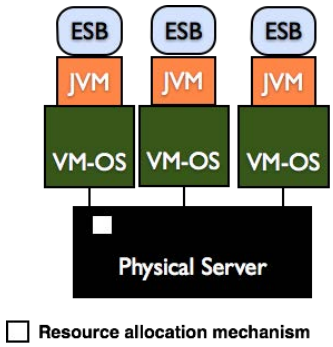


**Figure 3. 7 Vertical elasticity mechanism**

- a mechanism to allow tuning dynamically ESB components resources. Indeed, the ESB is composed of components; and computer resources (e.g. threads) can be allocated to them. For instance, a number of threads can be initially allocated to the main JBI components (BC, SE, NMR). These threads are used by the components when they perform their mediation roles. When all the threads are busy, the components will not be able to process more requests. Therefore, developed mechanism can be used to add at runtime the number of threads allocated to components and to avoid this kind of situation.

*Horizontal elasticity*

Cluster and federation deployment models are existing approaches that allow exploiting the distributed capabilities of ESB. Several mechanisms can be developed to apply the horizontal elasticity concept to manage clustered or federated ESB instances or components.

- **cluster management mechanisms**. A cluster consists in a set of ESB instances distributed on one or several computers and that act as a single virtual resource. It is generally associated to a load balancer component that distributes the incoming load across the instances.
    - a mechanism to add or retrieve new ESB instances to an existing cluster to allow ensuring a good QoS and scalability can be designed;
    - a failover mechanism can be applied so that if one node becomes unavailable, the other node(s) will continue to process the requests.
- **federation management mechanisms.** A federation is increasingly used for large enterprise integration. It allows the subdivision of a whole system into several domains. Involved systems and services of each domain can be interconnected by one ESB instance. The interconnection of the different domains is managed by interconnecting the different

ESB instances. By this way, processes and services can be distributed and some of them can migrate from one instance to another according to their performance.

- similarly to the cluster, a mechanism to add or retrieve new ESB instances to an existing federation to allow ensuring a good QoS and scalability can be designed;
- a mechanism can allow the migration of part of the mediation processes to the new instance or the sharing of distributed ESB instances contents. The figure 3.8 describes an illustration of the migration concept. For instance with two federated ESB instances ESB1 and ESB2, the ESB1 hosts three orchestration processes (BPEL1, BPEL2, BPEL3) and ESB2 hosts one (BPEL4). If scalability issues are detected on ESB1, developed mechanisms and services perform the migration of the process BPEL3 from the ESB1 to the ESB2 with an aim of load sharing;

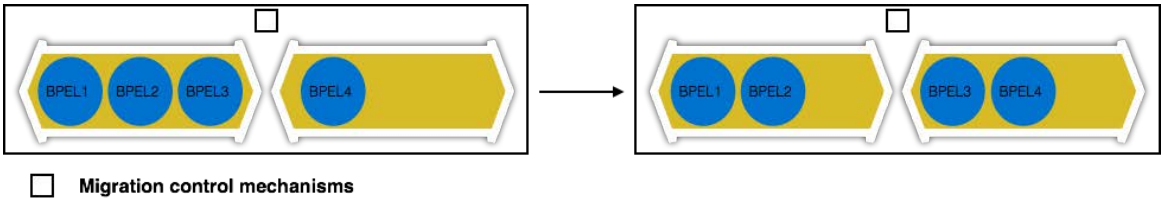


**Figure 3. 8 ESB level components migration**

- a mechanism can also be developed to support the migration of virtual machines between the physical servers, if it is the best strategy to apply to ensure scalability (Figure 3.9).

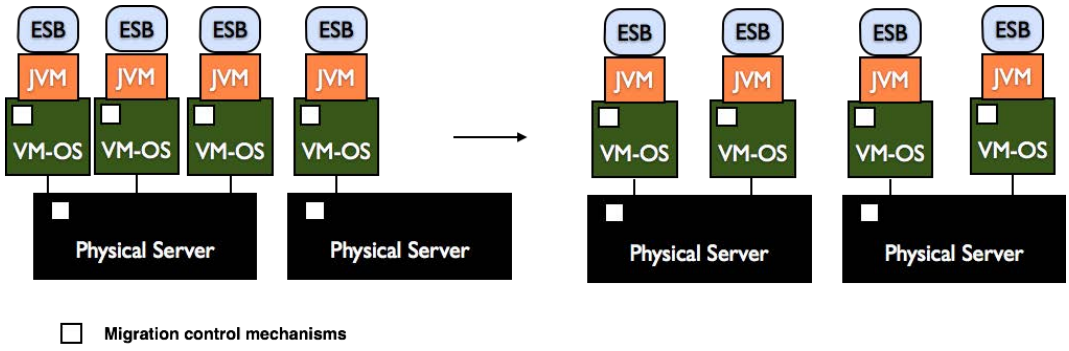


**Figure 3. 9 Virtual machines migration**

### 3.3.3 Summary

Our proposed approach to deal with QoS and scalability consists in the proposition of new mechanisms to extend ESB solutions for QoS and scalability management. These mechanisms

are presented in this section. The originality is that they are inspired from classical and enhanced QoS-oriented mechanisms proposed within network and transport protocols and on virtualization and cloud computing paradigm for an efficient management of computing resources and scalability.

The proposed mechanisms can be simple, and then can be directly used. For instance the partial reliability mechanism can be easily activated or deactivated with the percentage of requests to discard. They can be complex composed by several tasks that need to be done in sequence or in parallel. For instance to add an instance to a cluster it is needed to deploy a new ESB instance, to add the new instance to the cluster and to reconfigure the load balancer to take into account the new instance. These kinds of complex mechanisms can be simplified by subdividing their tasks in several sub-mechanisms that will be coordinated and orchestrated.

Next section presents the design of the architecture that allows managing the use and/or the coordination in an autonomic way of the proposed mechanisms to extend the ESB solutions for QoS and scalability management.

## 3.4 Autonomic Service Bus architectural framework

The architectural framework of the ASB proposed by following the MDA approach is described hereafter. We define expected functionalities of the ASB (CIM level) before presenting its components structure (PIM level) and an adaption specific to the JBI framework and a “cloud-oriented” infrastructure (PSM level).

### 3.4.1 Computation independent model of the ASB

The following paragraphs detail expected ASB’s functionalities and use cases to describe how it enhances a basic ESB solution.

A basic ESB solution supports synchronous and asynchronous one-to-one, one-to-many or many-to-many communications with the ability to link various services providers and consumers, based on their offers and requests. Its main functions are message routing between providers and consumers, reliable messages delivery, messages transformation, protocols adaptation, services composition and orchestration support, etc. These basic functionalities

are enhanced to have the ASB. Expected added extensions focus on the satisfaction of QoS, scalability, manageability and self-manageability requirements. They are introduced hereafter.

*ASB functionalities for QoS and scalability satisfaction*

The ASB solution participates in the management of QoS and scalability offered to consumers that invoke services accessible through the bus. In order to ensure this, the ASB uses a set of mechanisms available to deal with the QoS and scalability anomalies (e.g. a congestion control mechanism to avoid overload of the ESB components or communication channels). For that, a developer is able to easily extend the ESB implementation and the underlying computing resources by developing and deploying mechanisms for the QoS and scalability management.

The ASB solution shall deal with the complex management of scalability and QoS in the dynamic context of distributed systems. In order to ensure this, the ASB integrates an autonomic manager to control the state of the integrations and communications through the ESB, the state of the ESB components and the communication channels and finally the state of the computing resources on which the ESB instances are running. The autonomic manager collects monitoring data that will be analysed to predict or to detect contexts changes and anomalies (called “symptoms”). The autonomic manager uses policies and rules to decide about the adequate mechanisms to perform in the case of symptoms prediction and/or detection. For that, the ASB shall allow an administrator to define the symptoms that need to be detected and to define policies and rules that guide the autonomic behaviour.

The ASB also integrates models to characterize symptoms, corrective mechanisms and adaptive rules and polices. Semantic technologies based on ontologies can be followed and an ontology developer shall be able to define the different semantic models.

The figure 3.10 summarizes the ASB use cases.

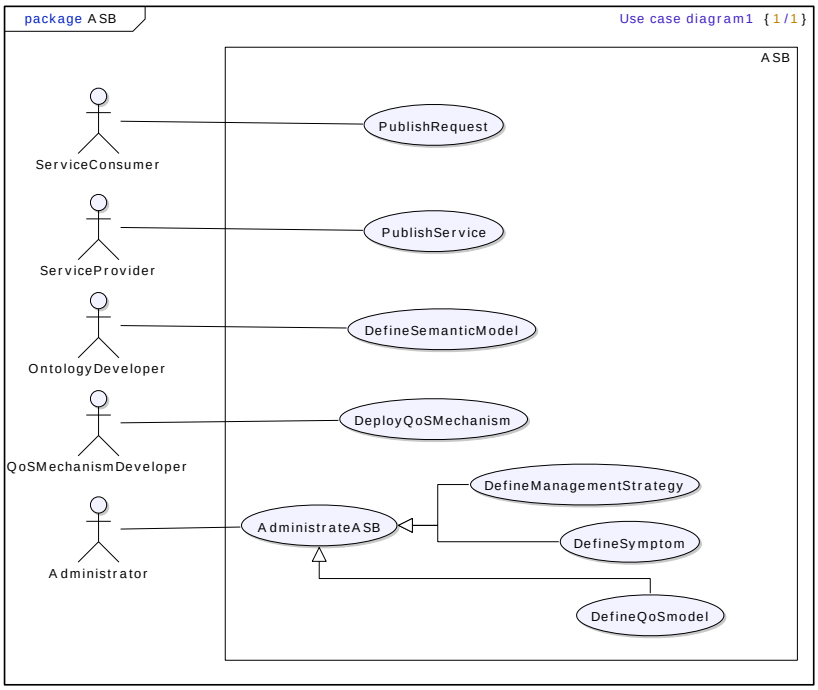


**Figure 3. 10 ASB Use Cases**

Next section gives the architectural design describing the structure of the ASB.

### 3.4.2 Platform independent model of the ASB

Following and implementing the IBM Autonomic Computing framework, the ASB can be seen as an autonomic element or a set of autonomic elements that result from the composition of managed elements (i.e. ESB instances, QoS and scalability oriented mechanisms, underlying network and computing resources, etc.) and an autonomic manager (i.e. a MAPE control loop) that guarantees the self-properties based on high-level policies.

Next figure 3.11 gives a global overview of the proposed ASB architecture. The fundamental managed element is the ESB, which offers services allowing efficient collaboration and mediation between distributed systems. The ESB can be deployed as a single instance, a cluster of several instances or a federation of distributed instances. In the case of a cluster or a federation deployment model, the clustered or federated instances are taken as the fundamental managed elements.

The ESB solutions are in general java-based implementations, running on a java virtual machine (JVM) deployed on a virtual or physical IT infrastructure. So to put in place an ASB,

the underlying computing resources (JVM and virtual or physical IT infrastructure) are also considered among the managed elements.

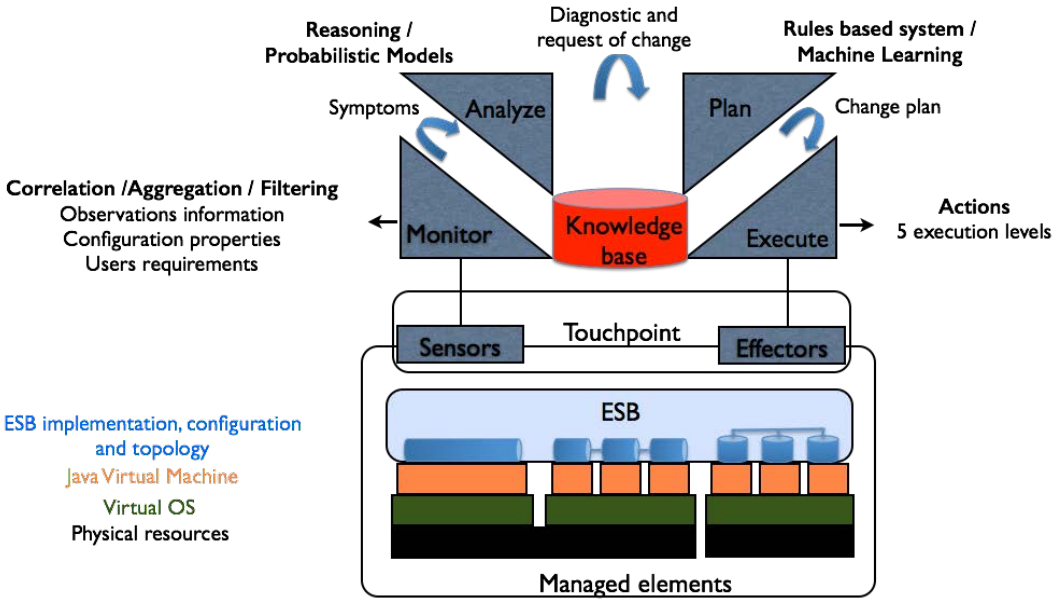


**Figure 3. 11 ASB Architecture Overview**

Several mechanisms are deployed and used for scalability and QoS management. They are configurable and considered among the managed elements.

All the managed elements (ESB instances, ESB components, computing resources, QoS and scalability mechanisms) offer the required touchpoint interfaces (sensors and effectors) in order to be controlled remotely.

An autonomic manager controls the managed elements by implementing the monitoring, analysis, plan, and execution functions. The Monitor component uses the sensors to get the state of the ESB instances, the java virtual machine, or the virtual or physical machines. The monitoring metrics are processed to detect symptoms. The Analyze component identifies the cause of the symptoms through a diagnostic and sends a request of change to the Plan component. The Plan component defines the change plan to apply. The Execute component applies the adaptive actions on the ESB instances, ESB components, computing resources and QoS and scalability mechanisms using the effectors.

The autonomic manager uses a knowledge base that contains the data providing a semantic description of the deployed services and mechanisms, but also a semantic description of the policies and rules to be applied in order to trigger actions implementing the self-management functions and the autonomic behaviour of the ASB.

The figure 3.12 illustrates the UML class diagram of the ASB composed of an autonomic manager and the managed elements.

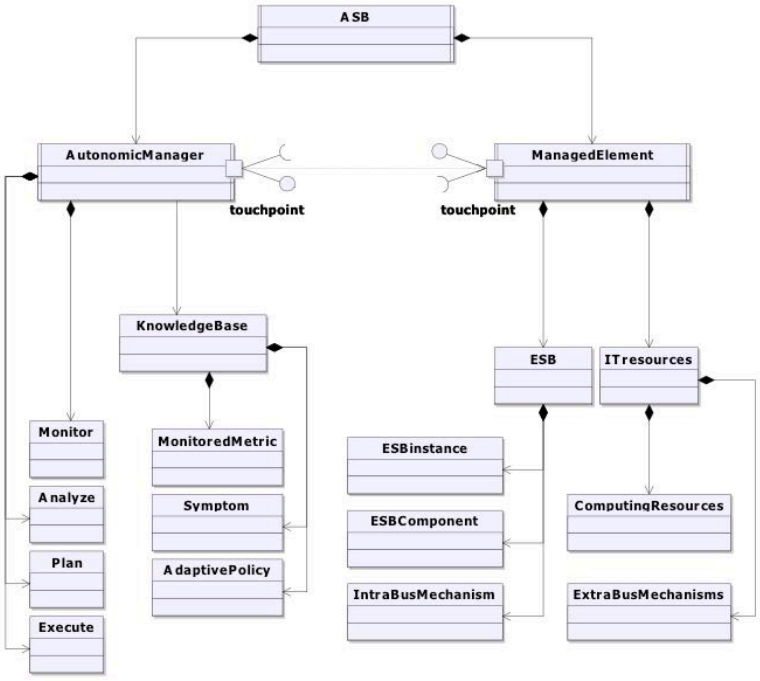


**Figure 3. 12 ASB UML Class Diagram**

The Figure 3.13 illustrates the UML composite structure diagram of the ASB resulting from the composition of the ESB and its underlying infrastructure coupled to an autonomic manager. The ASB offers three external interfaces:

- *serviceProvider* that allows a service provider to publish services;
- *serviceConsumer* that allows a service consumer to send requests;
- *admin* that allows an administrator to control the smart properties of the bus.

For simplicity concern, the administrator actor is also considered here as the QoS and scalability oriented mechanisms developer and ontology developer actors.

The ESB and the IT computing resources expose interfaces:

- *ESBsensor, ITresourcesSensor,* which provide monitoring operations;
- *ESBeffector, ITresourcesEffector,* which provide execution operations.

The autonomic manager is connected to the ESB and the IT computing resources through these interfaces. The different components that perform the control loop have specific interfaces to communicate between them and to interact with the knowledge base.

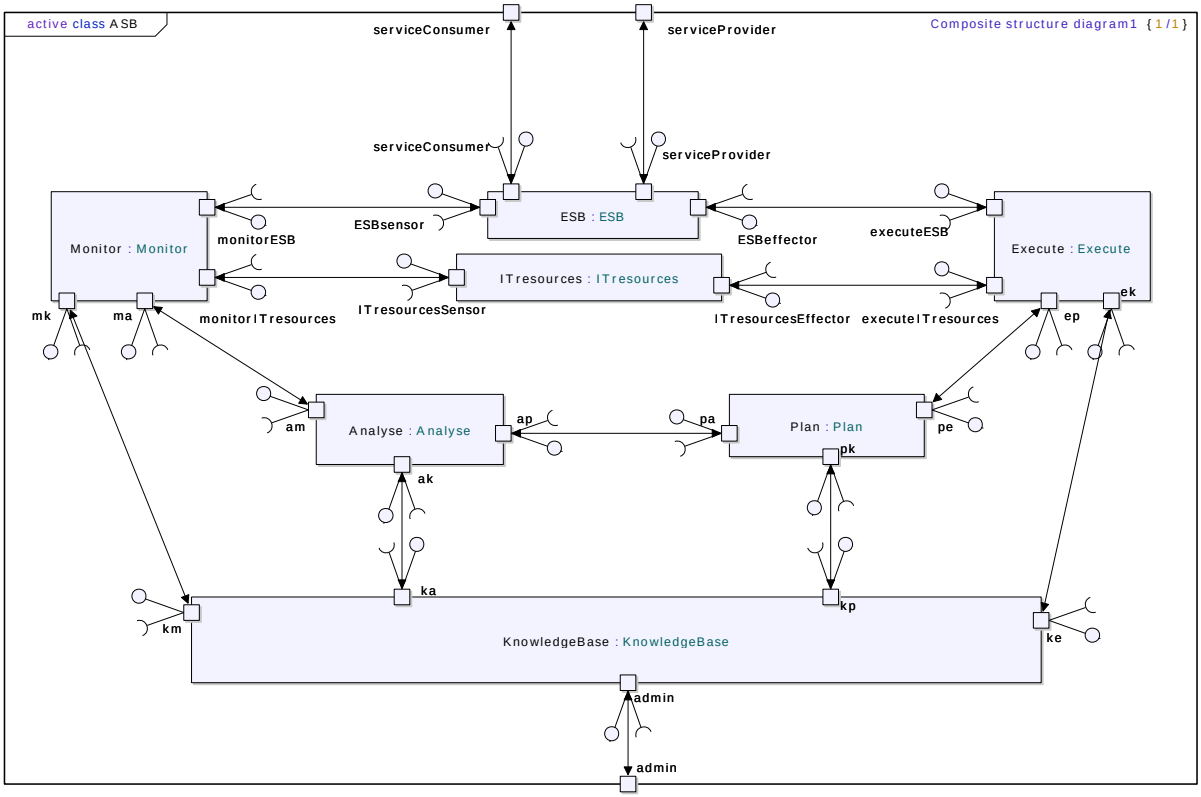


**Figure 3. 13 ASB UML Composite Diagram**

The figure 3.14 presents an example of sequence diagram that illustrates how the ASB works.

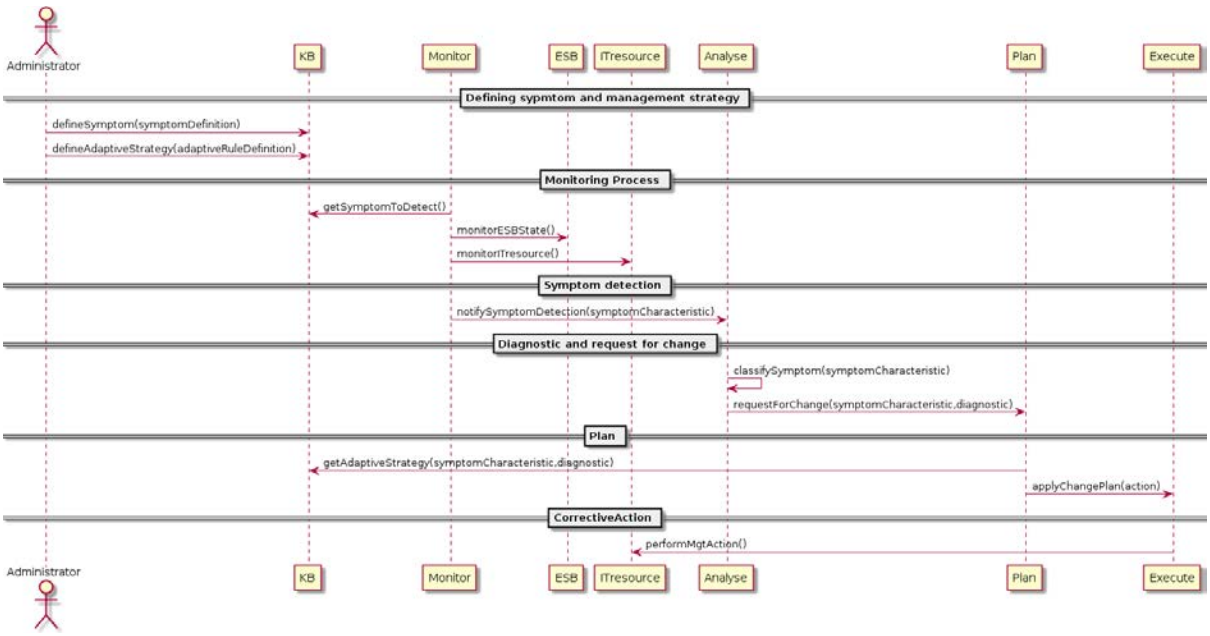


**Figure 3. 14 ASB UML Sequence Diagram**

Next section gives an adaption of the ASB framework specific to the JBI framework and the cloud computing deployment context.

### 3.4.3 Platform specific model of the ASB

Following and implementing the IBM Autonomic Computing framework, the ASB has been designed in a generic way to be implemented with any existing ESB implementations and deployment configurations. However, in this thesis, the Java Business Integrator (JBI) specification and the cloud computing paradigm have been chosen for a flexible and extensible solution that allow the easy introduction of mechanisms for QoS and scalability requirements. The goal of this section is to present how the proposed architecture can be implemented using a JBI-compliant ESB solution deployed on a "cloud-oriented" infrastructure (Figure 3.15).

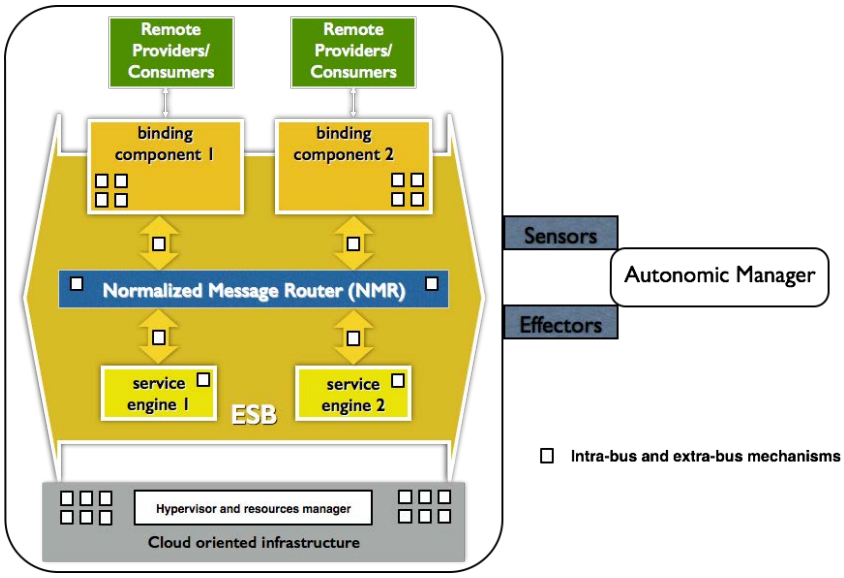


**Figure 3. 15 ASB JBI compliant architecture**

By using a JBI-compliant ESB, intra-bus mechanisms can be easily implemented by extending traditional components of the ESB (Binding Component, Delivery Channel, Normalized Message Router, Service Engine). For instance normalized message router can be extended to implement a priority driven mediation process. Extended components can easily be added to the implementation as a plug and play component. Next figure 3.16 details how to implement the flow control mechanism by exploiting the JBI framework characteristics. The binding component used by the consumer (inbound binding component –*IBC*–) and the binding component used by the provider (outbound binding component –*OBC*–) are extended

to modify their traditional behaviour. The *IBC* sends consumer requests. The Outbound Binding Component (*OBC*) puts the messages in a buffer while the service provider extracts them. By this way, the *OBC* can monitor the load and the performance of the provider (e.g. the number of buffered messages waiting to be processed by the provider). When this number reaches a given value, for instance the maximum capacity of the provider's buffer, the *OBC* notifies the *IBC* about the potential provider saturation. Receiving this information, the *IBC* reduces the sending rate or refuses more incoming requests.

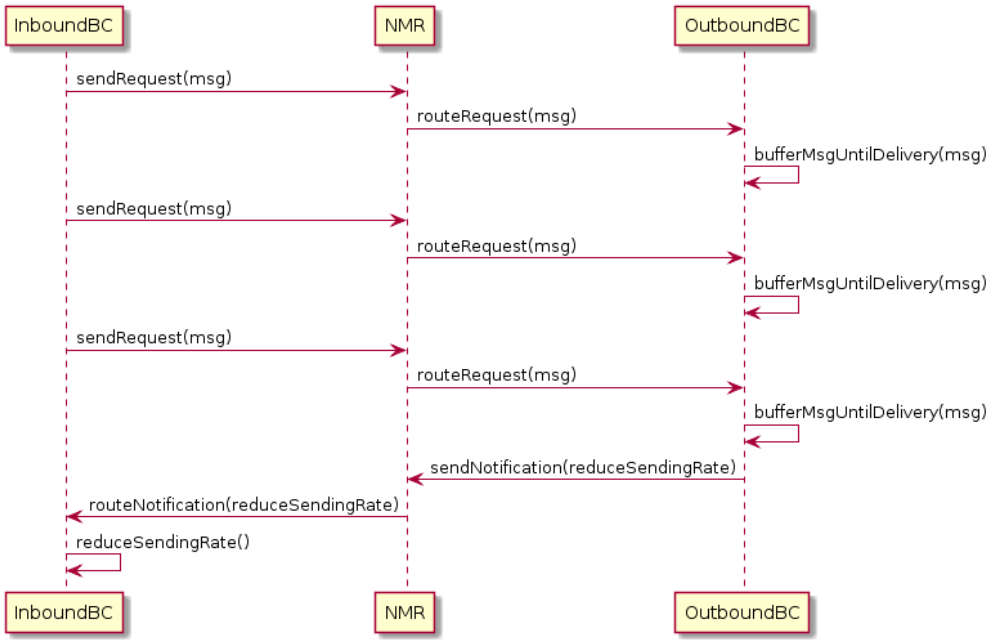


**Figure 3. 16 Flow control mechanism sequence diagram**

The "cloud-oriented" virtual infrastructure can be extended with solutions that exploit the hypervisor functionalities to implement the extra-bus mechanisms. Next figure 3.17 illustrates the vertical elasticity concept applied to a virtual machine that hosts the ESB. The mechanism exploits the hypervisor interface to get the state of the physical server. If the physical server has enough resources, more CPU can be allocated to the virtual machine.

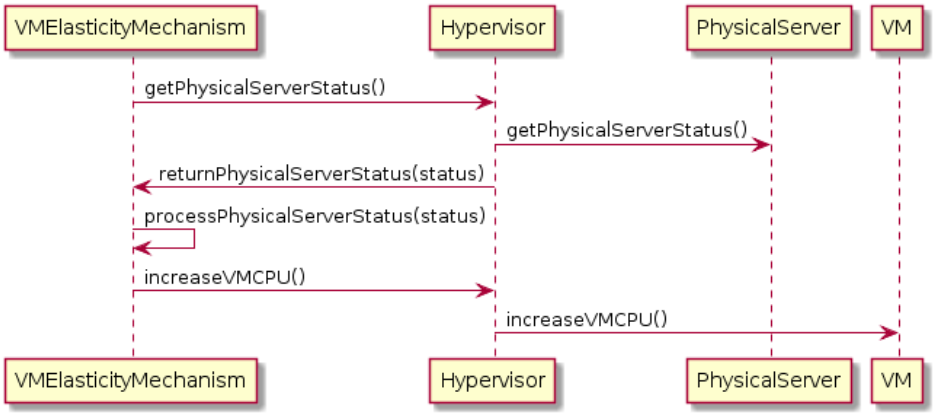


**Figure 3. 17 Virtual machine elasticity mechanism**

### 3.4.4 Summary

The architectural framework of the ASB is detailed in the previous section. We follow the MDA approach to define expected functionalities of the ASB (CIM level) before presenting its components structure (PIM level) and a specific JBI compliant solution deployed on a "cloud-oriented" infrastructure (PSM level).

The proposed architecture can be easily extended and made even more complete by integrating other existing middleware platforms (as the MOM solution) that will help to better implement proposed mechanisms for QoS and scalability satisfaction.

Next section introduces the platform used to go towards an implementation of the proposed architecture.

## 3.5 Towards the ASB framework implementation

To implement an Autonomic Service Bus and to achieve an efficient and controlled QoS and scalability of systems integration, an Emulation Platform for ESB Systems (EPES) is used. The design of this emulation platform is presented hereafter.

### 3.5.1 Specification of EPES

EPES is an emulation platform intended to allow simulating the behaviour of distributed systems (services consumers and providers) that communicate through a real ESB implementation. EPES allow also assessing ESB systems by identifying and characterizing their limits. In the ASB context, it can be used by:

- an administrator, who manages an ESB and needs to assess its deployment to avoid overestimating or underestimating the capacity of the used ESB;
- a designer of QoS and scalability oriented mechanisms, who needs to test and validate proposed extensions;
- an administrator who has to build the knowledge models of an ASB implementation.

EPES can also be useful in other contexts, for instance for users who want to compare ESB implementations.

To exploit EPES functionalities, the different users have to write the description of the scenario they want to run. For the scenario description, a set of parameters relative to the ESB configuration but also the behaviour of the service consumers and providers can be defined.

Being able to configure all of these parameters for a scenario makes EPES generic and allows running any type of processes and scenarios. Moreover, EPES is designed to facilitate its extensions. New ESB implementations, mechanisms and solutions can be easily integrated and evaluated. Furthermore, it is ready to respond to the dynamic increase of simulated consumers and providers while still preserving an acceptable level of performance (running time, reliability of results, etc.).

### 3.5.2 Design of EPES

The purpose of this section is to present EPES design. Based on defined specifications in the previous section, the different components of the system are (Figure 3.18):

- a *Controller*. Its main goal is to parse the scenario and to configure the consumers and providers that will communicate through the ESB;
- a *ConsumerAgent*. It implements the consumer web services. They are configurable by the controller and invoke the provider through the ESB;
- a *ProviderAgent*. It implements the provider web services. They are configurable by the controller and can be invoked by the consumer;
- an *ESB*. The ESB is a real implementation that supports the communication between consumers and providers.

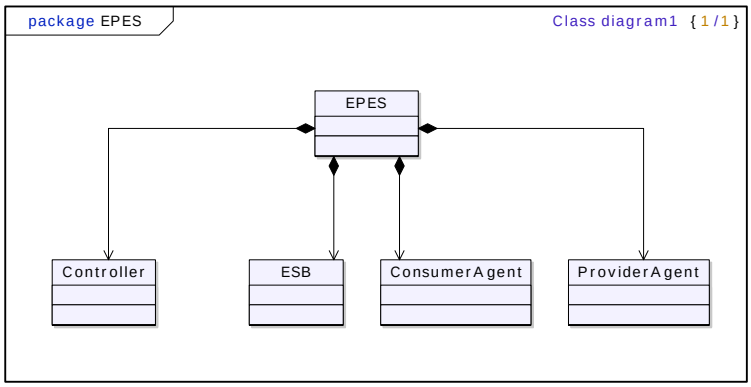


**Figure 3. 18 EPES UML class diagram**

The figure 3.19 gives the composite diagram of the EPES.

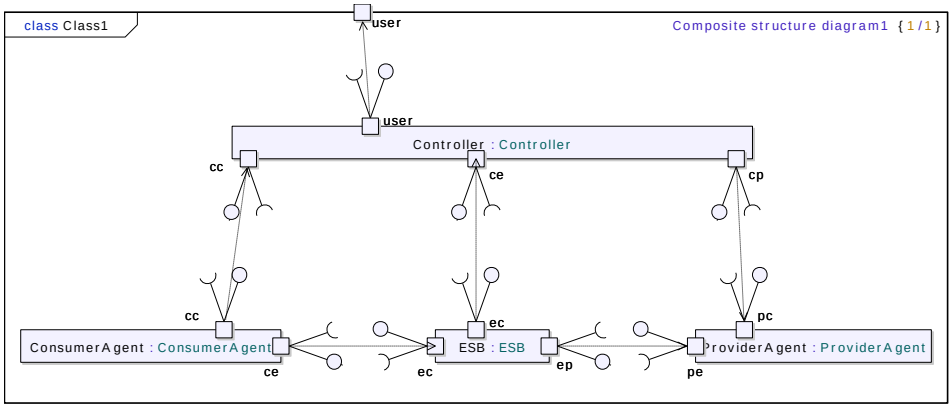


**Figure 3. 19 EPES UML structure diagram**

The technical solutions for the EPES implementation are detailed in the next section.

### 3.5.3 Implementation of EPES

The implemented version of EPES is based on a JBI compliant ESB implementation. We use the OpenESB implementation version 2.3. However the ESB can be easily replaced by any ESB.

The deployment is done on a flexible “cloud-oriented” virtual infrastructure. The chosen virtualization solution is Proxmox Virtual Environment [PRO 14]. Proxmox allows deploying and managing virtual machines used to host the OpenESB instances. It also allows us to dynamically configure resources (memory, processor, hard disk) to be allocated to the virtual machines. Each virtual machine is based on OpenVZ open standards with Linux Ubuntu 12.04 LTS operating system (OS), a configurable QEMU Virtual CPU.

The consumers and providers agents are a set of web services developed using the java programming language based on the JDK version 6. These web services are deployed on the glassfish application server version 3.1.

### 3.5.4 EPES and ASB

The implemented version of EPES is the starting point of the ASB implementation (Figure 3.20). It allows:

- conceiving the different services and solutions that implement the autonomic manager components (Monitor, Analyze, Plan, Execute components);
- knowing and characterizing QoS and scalability degradations that may arise when an ESB is used;
- deploying, testing and validating efficient extensions and mechanisms to deal with degradation;
- building the knowledge base that will be used to guide the autonomic behaviour (e.g. diagnostic models, association of issues and specific corrective actions, etc.).

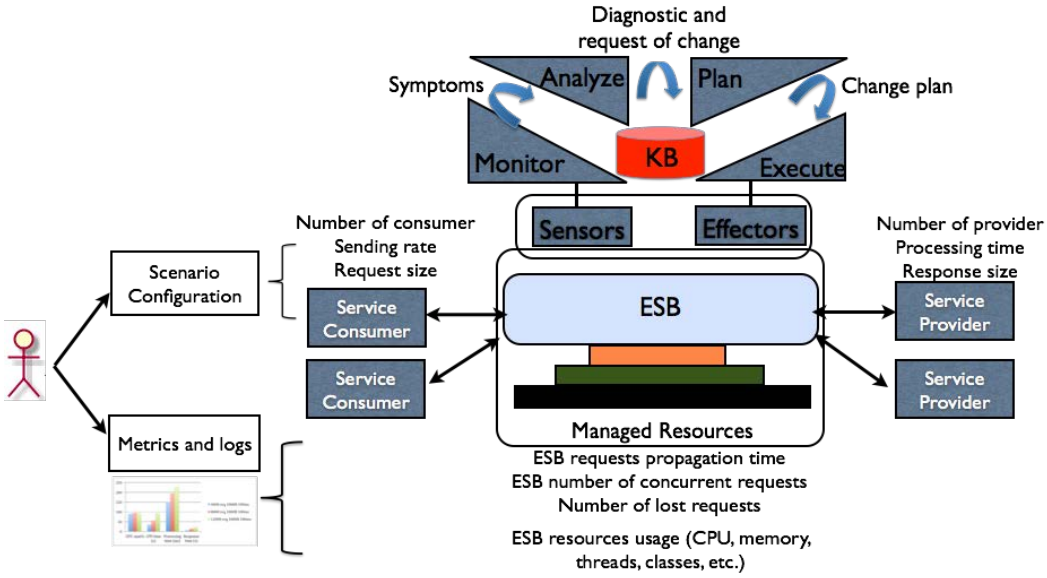


**Figure 3. 20 Usage of EPES in the ASB context**

## 3.6 Conclusion

In this chapter, the design of several mechanisms that can be added to an ESB to enhance the global scalability and QoS was introduced. The originality of the proposed mechanisms is that they are inspired from QoS-oriented functions and mechanisms used in the Internet at the

transport and network OSI layers, such as congestion control, traffic shaping, differentiation among users or error control. The goal is, for instance, to react in front of congestions that can occur both on service providers and on the internal bus components. Likewise, differentiated or guaranteed QoS-oriented models can be implemented by mechanisms able to manage priorities between service consumers or providers if needed. Cloud-oriented mechanisms are also proposed to control the ESB topology deployment model and the underlying computing resources.

The overview of the ASB architecture has been also presented. The different actors and the ASB functions have been identified. Efficient design approaches have been followed to provide a generic architecture applicable to any kinds of ESB implementations.

The architecture results in the extension of a standard ESB specification and its underlying computing resources with the proposed QoS and scalability mechanisms and an autonomic manager designed to manage the coordination and use of the mechanisms in a smart and efficient way. It can be improved by integrating other existing middleware platforms (as the MOM solution) that will help to better implement proposed mechanisms QoS and scalability satisfaction.

An Emulation Platform for ESB Systems (EPES) has been proposed to allow the implementation of the architecture by following the transition steps identified with the five levels of autonomic maturity proposed IBM for an incremental implementation starting from a basic and manual ESB and targeting an autonomic ESB solution. EPES is based on a JBI compliant ESB implementation deployed on a “cloud-oriented” infrastructure. Therefore, it is extensible and allows us to go towards an implementation of the ASB.

Next two chapters present how we follow these iterative phases to have an implementation of the proposed ASB.

The chapter 4 entitled “Monitoring” presents a monitoring solution for detecting QoS and scalability issues. The solution is based on monitoring services proposed to supervise the different elements that need to be observed and an event-processing module proposed for the treatment of all the monitored data.

The chapter 5 entitled “Analysis, Plan and Execution” presents solutions developed to build a knowledge base needed for the interpretation of monitored data and symptoms, and for the definition of the adequate plan to execute. A first section presents the development of extensible models for a reliable and smart analysis of symptoms. The models are designed and used to diagnose symptoms. A second section presents the development of models and rule-based systems that guide the choice of the adaptive plans to deploy when anomalies are detected. The models are used to correlate a diagnostic to a corrective solution. The final part of this chapter 5 details the execution services aimed at automatizing the use and coordination of mechanisms proposed in order to satisfy the scalability and the QoS demands.

# Chapter 4. Monitoring

**Contents**



## 4.1 Introduction

Based on the architectural design and the followed incremental methodology to implement an autonomic system presented in the previous chapter, the first step for developing the Autonomic Service Bus (ASB) consists in dealing with the manageability by defining how the system in operation can be monitored and controlled [NOR 13]. At this managed maturity level, a human administrator is needed to analyze detected symptoms and to initiate the actions to be executed (Figure 4.1).

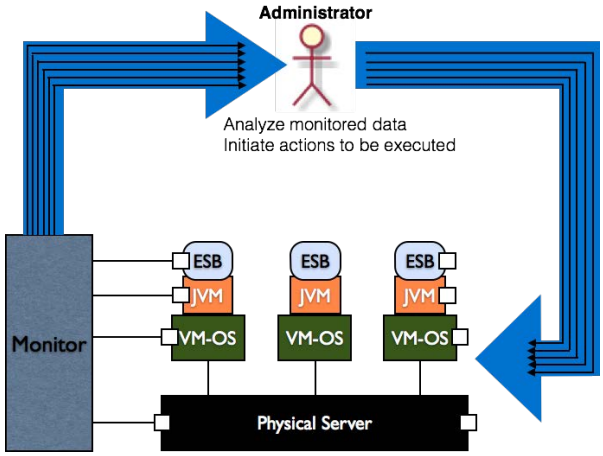


**Figure 4. 1 Managed maturity level**

The goal of this chapter is to detail our proposed solutions for the monitoring solution of the ASB.

Monitoring is the fact of measuring, supervising and getting information and metrics about a system and its context. It allows detecting symptoms such as QoS or scalability degradation and will help to diagnose their causes (e.g. lack of resources, increase of demands, etc.).

By considering a JBI compliant ESB implementation deployed on a “cloud-oriented” infrastructure, several parameters have to be monitored because they can impact the QoS and scalability satisfaction namely:

- the configuration and state of ESB components and communication channels but also of supported services and processes. For instance, when the ESB has to integrate a large number of parallel and concurrent systems, overload situations of the router component can appear and create message losses;
- the configuration and state of the java virtual machine (JVM). For instance, when the ESB has to integrate a large number of parallel and concurrent systems, the memory offered by the JVM can be limited to support ESB tasks and communications;
- the configuration and state of the physical server and virtual machines that host the ESB. Indeed, limitations of QoS and scalability of an ESB can be achieved when it uses all the resources of the host system [UEN 06].

Therefore, a monitoring solution for the management of QoS or scalability asks for:

- the supervision of each operational level;
- the processing of collected data to calculate an average (e.g. average response time) and to compare it with a defined threshold in order to detect potential symptoms. Additionally, the correlation (link between the metrics), aggregation (group different level metrics) and filtering (selection of specific patterns) of collected data are needed to identify or predict more complex patterns that can be modal, temporal, etc. such as a sequence of response time values exceeding a defined threshold and following an increasing tendency;
- a lightweight monitoring approach avoiding a negative impact on the QoS and scalability of the ESB. Indeed, the number of monitored parameters and the quantity of collected data can quickly increase. In this context, monitoring activities could consume internal resources or have a negative impact on the normal operations (e.g. by delaying the messages).

To satisfy these requirements, a set of monitoring specifications, standards and tools have been proposed in the literature. Some of them are Java Management eXtension (JMX) standardized in 2003 for monitoring and managing java-based applications [JSR-206], Simple Network Management Protocol (SNMP) [RFC 1157] widely used for monitoring and detecting anomalies on network devices and systems (e.g. switch, router, server, etc.), Nagios [NAG 14] for the supervision of applications, servers, network devices, business processes, etc., or again Opsview [OPS 14] for the monitoring of network, physical, virtual and cloud-based servers, application server, etc.

Regarding the correlation, aggregation and filtering of monitored data to identify or predict complex patterns, event-processing technologies are more and more used for tracking exceptional events or situations that can appear on various components [ETZ 10]. Business Activity Monitoring (BAM) solutions are also introduced in order to gather, aggregate, analyze, correlate and present monitoring data that can come from many levels of heterogeneous enterprise systems [MCC 02]. BAM solutions are generally based on event-processing technologies [PSI 12].

More specifically to ESB solutions, [YUA 08] follows JMX standards to propose a service-monitoring framework for ESB-based services that allows getting more information related to hosted services (number of invocations, results, response time, etc.) and processes (e.g. number of exchanged messages between a BPEL engine and services). The proposal is designed to monitor various parameters while limiting the overhead. However the proposed solution does not integrate neither the monitoring of the used underlying computing resources parameters (that can have an impact on QoS and scalability of the hosted ESB), nor a module for the detection or prediction of complex patterns (symptoms). In [PSI 12], authors propose a monitoring framework for ESB with a set of monitoring mechanisms that exploit JBI monitoring capabilities. The proposed solution is also based on JMX and allows getting information related to the qualitative ESB's parameters (e.g. uptime), exchanged messages (e.g. sent and received requests, sent and received replies, etc.), hosted services (invocations timestamps, response time, etc.) and processes performance. Similarly to [YUA 08], authors of [PSI 12] do not integrate the monitoring of the underlying computing resources that can impact the QoS and scalability of the ESB. However, they exploit event-processing technology functionalities for the detection or prediction of complex symptoms.

In contrast to these works, a complete Monitor component is proposed in this thesis, which is based on:

- several monitoring services proposed to supervise not only the ESB level, but also the underlying infrastructure resources level (e.g. Java Virtual Machine, virtual machines and physical server on which run the ESB, etc.). Different approaches to develop these monitoring services for the different levels are studied to measure their cost and to choose the ones in order to limit the impact on the ESB and the distributed systems operations;
- an event-processing module included in addition to the monitoring services for the treatment of the monitored data in order to recognize and detect complex patterns correlated to potential symptoms. Methodologies to identify, define and characterize symptoms that need to be detected by the event-processing module are also proposed.

The chapter is structured as follows: section 4.2 presents the monitoring services to observe and supervise the different operational levels. Section 4.3 presents the event-processing module for the treatment of the monitored data and the detection of symptoms. Section 4.4 summarizes and concludes the chapter. Approaches and solutions to make the Monitor component more smart and efficient are discussed.

## 4.2 Multi-levels monitoring services

As already introduced, tasks of the proposed Monitor component are distributed in two steps (Figure 4.2):

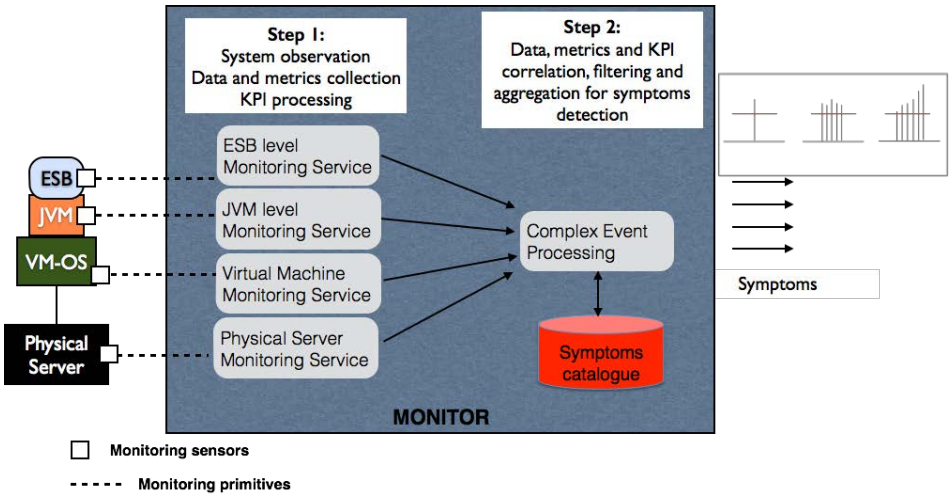


**Figure 4. 2 Monitor component structure**

- the first step is based on services to observe and supervise all the defined managed elements (ESB level monitoring service, JVM level monitoring service, virtual machine monitoring service and physical server monitoring service, etc.) and to collect the data and metrics to be sent to the second step afterwards.
- the second step is based on a complex event processing solution to aggregate, filter and correlate the collected data and metrics in order to detect symptoms. A symptom catalogue (database in red) is proposed to guide the complex event processing rules. This database belongs to the knowledge base.

Next sub-sections present developed services to monitor the ESB level, the Java Virtual Machine level and the virtual and physical infrastructure level.

### 4.2.1 ESB level monitoring service

Two main approaches are followed to develop the monitoring services for data collection namely [LIG 13]:

- active monitoring or polling, which is an approach that consists in the capacity to simulate the monitored system's users behaviour by sending dummy and time-stamped requests. The observation of these injected requests will give the state of the system. The active monitoring may impact the result since additional traffic that adds an overload to the system and that uses resources of the monitored system is injected;
- passive monitoring or (wire) trap, which is an approach that consists in the capacity to report the state of the monitored system based on the observation of events going through it. The passive monitoring measures the real performance since it does not add overhead to the system.

These active and passive monitoring approaches are followed to explore three ways to develop ESB level monitoring service.

**1)** The figure 4.3 presents the first (active) approach in which the ESB level monitoring service is based on two entities that interact through the ESB using dummy messages with timestamps.

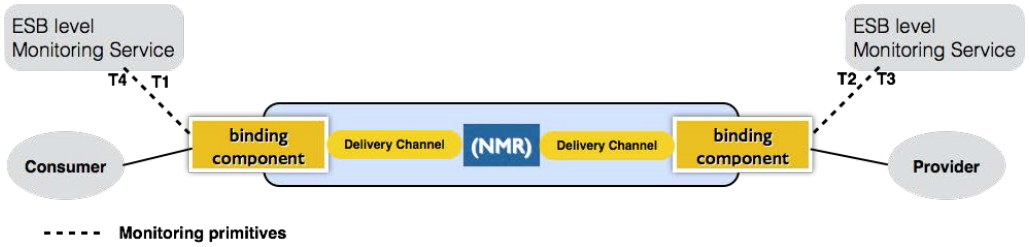


**Figure 4. 3 ESB level active monitoring service**

The sequence diagram of the figure 4.4 presents a behavioural example of these two entities (*MonitorServiceSender* and *MonitorServiceReceiver)*. Based on a defined frequency, the *MonitorServiceSender* sends dummy requests with a timestamp **(T1)** to the *MonitorServiceReceiver* through the ESB. The *MonitorServiceReceiver* enriches the dummy requests with the reception date **(T2)** and the date it sent the responses **(T3)**. At the reception of the responses, the *MonitorServiceSender* get the date (**T4**) and uses all these timestamps to evaluate the current performance of the ESB. For instance the equation 1 (eq 1) gives the response time from the consumer point of view (RTT) and the equation 2 (eq 2) represents the time spent by message through the ESB (ProT).

$$\text{RTT} = T4 - T1 \quad \text{(eq 1)}$$

$$\text{ProT} = (T2 - T1) + (T4 - T3) \quad \text{(eq 2)}$$

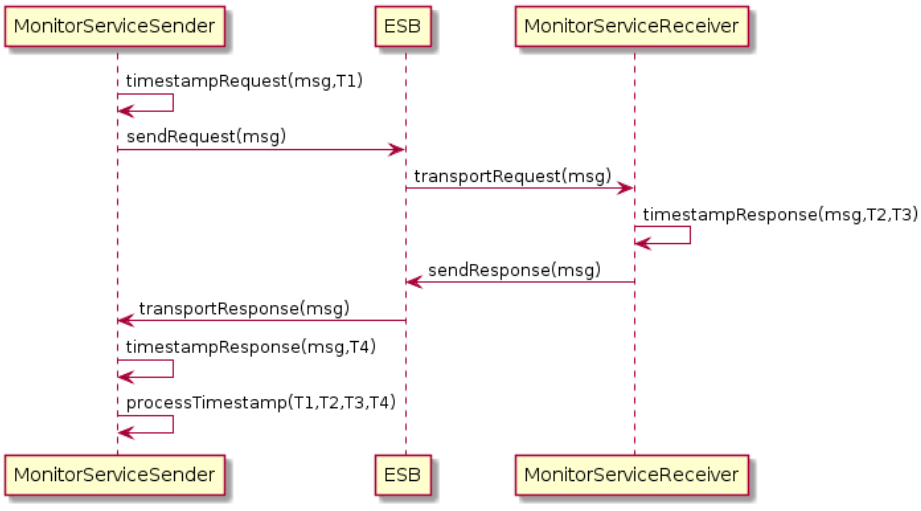


**Figure 4. 4 ESB level active monitoring service behaviour**

If the *MonitorServiceSender* and *MonitorServiceReceiver* entities are deployed on the same machine as the ESB, this method is simple and efficient even if it adds an overhead since additional traffic is injected. However, if the entities are deployed on another machine, this approach becomes complex since the network propagation time between the monitoring

entities and the ESB needs to be taken into account. For instance, the way to calculate the time spent by a message through the ESB changes. In the equation 3 (eq 3), we assume that the *MonitorServiceSender* and the *MonitorServiceReceiver* are located on the same machine and the time between them and the ESB (deployed on another machine) is called NetP.

$$\text{ProT} = (T2 - \text{T1}) + (\text{T4} - \text{T3}) - (4 * \text{NetP}) \quad \text{(eq 3)}$$

This active monitoring service also allows getting the current state of the ESB. For instance, the sequence diagram of the figure 4.5 shows how it can be used to test whether the ESB is able to accept more incoming requests or not.

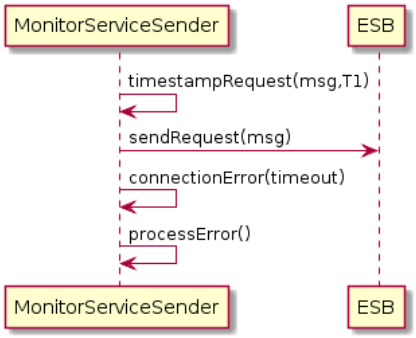


**Figure 4. 5 ESB level active monitoring service usage for connection error detection**

**2)** The next figure 4.6 gives the second approach explored to develop the ESB level monitoring service based on the extension of the binding components that are the endpoints for ingoing and outgoing requests. Indeed, the binding components can be extended to create a log, each time they receive or deliver a request. These logs are used by the monitoring service to evaluate the performance of the ESB (e.g. propagation time of a message through the ESB).

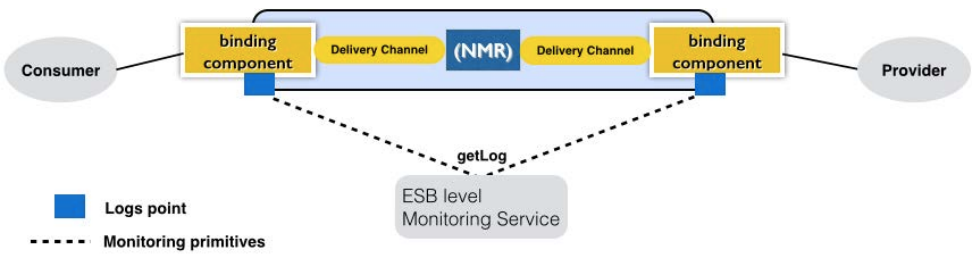


**Figure 4. 6 ESB level binding components logs approach**

The figure 4.7 illustrates the different interactions. When a *ServiceConsumer* sends a request, the *InboundBC* creates a log (e.g. requestReceptionTime) and sends the message to the *NMR*.

The *NMR* routes the message to the *OutboundBC*. The *OutboundBC* delivers the message to the *ServiceProvider* and creates a log (e.g. requestDeliveryTime). The *MonitorService* is able to process the logs to get the state of the ESB.

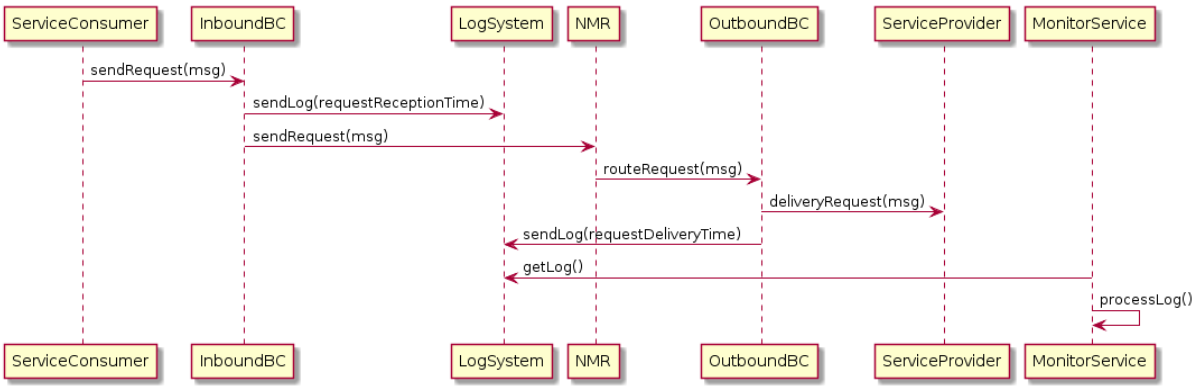


**Figure 4. 7 Components interaction of the ESB level binding components logs approach**

With this second approach, the real performance of the communication through the ESB is obtained since dummy traffic is not injected. However, it includes a high cost on the communication in terms of delay (*response time and propagation time*) since the binding components stop the messages to create the log before releasing them. From the consumer point of view, the RTT is extended by the time taken to manage the logs (Equation 4, eq4).

To solve this issue by avoiding or limiting the high overhead, more efficient binding components able to support parallel tasks are needed.

$$RTT_{final} = RTT_{initial} + delay\ to\ manage\ logs \quad \text{(eq 4)}$$

**3)** By exploring the JMX capabilities of the ESB, a third monitoring solution has also been developed. This solution is more efficient and scalable since dummy requests are not injected and the approach does not have an impact on the real communications.

The approach is based on a monitoring service able to observe and monitor the ESB framework and components without sending dummy requests or interrupting the communication.

The ESB components are taken as a set of managed beans (MBeans) able to inspect messages that go through them and to log statistics. They offer methods to get these statistics related to their state (Figure 4.8). We develop a service able to invoke periodically these MBeans to get the statistical data. The process of this data gives the state of the ESB.

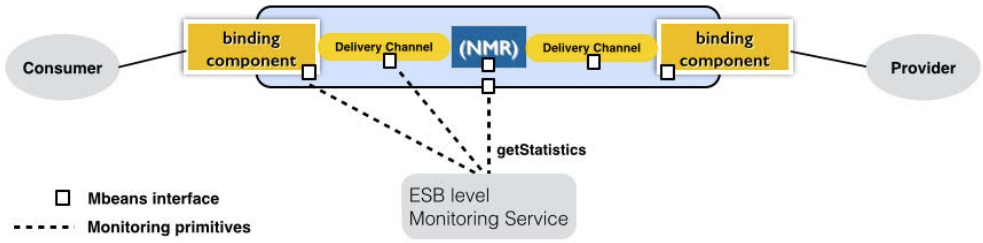


**Figure 4. 8 ESB level JMX-based monitoring approach**

The interface between the developed service and the MBeans is composed of operations that give statistics related to:

- the state of the ESB (stopped, started), the state of a specific binding component or service engine (*getServerState, getBCState, getSEState*);
- the average time spent by requests on the different components (*getDCAverageTime, getBCAverageTime, getNMRAverageTime, getSEAverageTime*);
- the throughput of each component (*getDCthroughput, getBCthroughput, getNMRthroughput, getSEthroughput*), the number of received requests by each component (*getDCNumberOfRequest, getBCNumberOfRequest, getNMRNumberOfRequest, getSENumberOfRequest*);
- the available resource for each specific binding component or service engine as the number of threads (*getBCthread, getSEthread*).

Compared to the second approach, the ESB components taken as MBeans are able to inspect messages that go through them and to log statistics in a cache without inferring the communication. This third approach is then more efficient and is mainly followed in the literature to provide ESB level monitoring solution [YUA 08] [PSI 12]. It guides the studied and proposed approaches for the JVM, physical server and virtual machines levels monitoring services presented in next sections.

### 4.2.2 JVM level monitoring service

ESB runs on a Java Virtual Machine (JVM) that provides the execution platform required by java-based applications. The JVM is an abstract computing machine with all the required resources (threads, memory, CPU, classes) to support java programs [LIN 13] [VEN 98]. The figure 4.9 gives an architectural overview of the JVM. It can support many threads loaded for the execution of the java code on the execution engine. This execution engine requires

memory areas. Some of the memory areas are dedicated to specific threads (program counter register, java stack). The others are shared among all the running threads (heap area, method area, native method stack).

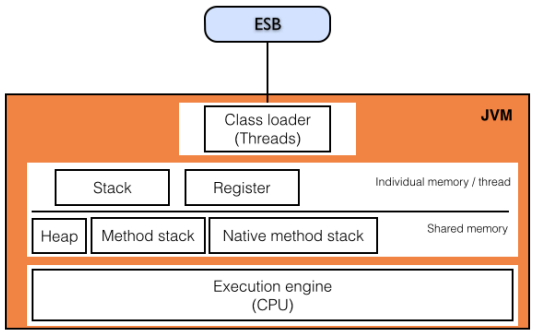


**Figure 4. 9 JVM structure**

If the JVM does not have enough resource to support the ESB tasks and the communications going through it, several errors will occur (e.g. *StackOverflowError* when a larger java stack is required, *OutOfMemoryError* when the heap memory is not enough or when the available memory does not allow to create a new thread, etc.) [LIN 13]. The main goal of the proposed JVM level monitoring service is to observe the JVM and detect or predict the occurrence of these issues. We decide to follow the passive approach without comparing several approaches (already done in the previous section). A monitoring service (similar to the JMX approach) based on two components is proposed. One component is deployed on the JVM side as a sensor that gives the JVM state when it is invoked. The second component is able to invoke the deployed sensor and to collect the different metrics (Figure 4.10).

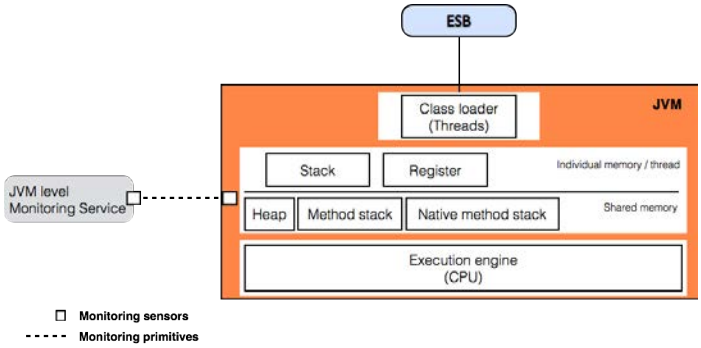


**Figure 4. 10 JVM level monitoring service**

Following operations compose the interface between the two components:

- *getClassNumber* that gives the number of classes loaded into the JVM;
- *getThreadNumber* that gives the number of threads used by the JVM to execute different instructions;

- *getMemoryState* that allows to collect the data related to the memory consumption;
- *getProcessLoad* that gives the number of processes running on the execution engine;
- *getNumberOfProcessor* that gives the number of processors that can be used by the execution engine. This parameter depends on the virtual or physical host machine;
- *getCpuState* that allows knowing how the JVM uses the CPU of the host machine.

Next section presents the physical server and virtual machines levels monitoring services.

### 4.2.3 Physical server and virtual machines level monitoring service

In this thesis, we consider a "cloud-oriented" deployment context based on a flexible virtual infrastructure to take advantage of the scalability and the rapid elasticity concept (horizontal or vertical scaling) related to the deployment of virtual machines or the tuning capacity of resources allocated to these virtual machines. Thanks to this flexible virtual infrastructure, virtual machines can be deployed on a physical server to host ESB instances. Resources (e.g. CPU, memory, etc.) of virtual machines can be provisioned or released according to the traffic through the ESB. Moreover, new virtual machines can be automatically deployed (or released) based on expected QoS and dynamic demands. All these operations require the monitoring of the virtual machines and the physical server on which these virtual machines are deployed.

In this context, we develop services to monitor the physical server and the deployed virtual machines. Similarly to the JVM level monitoring service, these services apply a passive monitoring (without comparing several approaches) based on two components. One component is deployed on the server side as a sensor that gives the state of the server and the virtual machines if it is invoked. The second component is able to collect the different metrics (Figure 4.11).

The most interesting operations to get the physical server and virtual machines states are:

- *getServerState* to retrieve the server state (stopped, started);
- *getServerCpuState* to have the CPU load of the server hosting the virtual machines;
- *getServerMemoryState* to get the memory state of the server hosting the virtual machines;
- *getServerDiskState* to know the hard disk state of the server hosting the virtual machines;
- *getVMState* to retrieve the virtual machines state (stopped, started);

- *getVmCpuState* to get the CPU load of the virtual machines;
- *getVmMemoryState* to know the memory state of the virtual machines;
- *getVmDiskState* to have the hard disk state of the virtual machines.

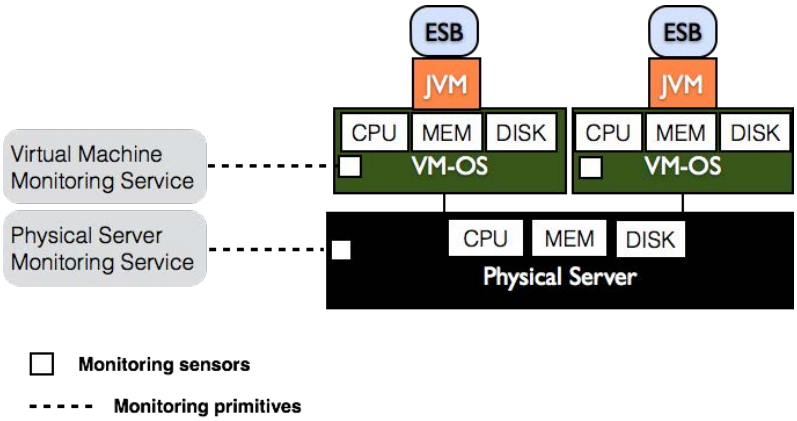


**Figure 4. 11 Physical Server and virtual machines levels monitoring services**

### 4.2.4 Summary

The different monitoring services presented in this section are the first part of the Monitor component of the ASB. They allow collecting all the data and metrics that give the state of the ESB and its underlying computing resources (Figure 4.12). Their evaluation and validation are presented in Chapter 6.

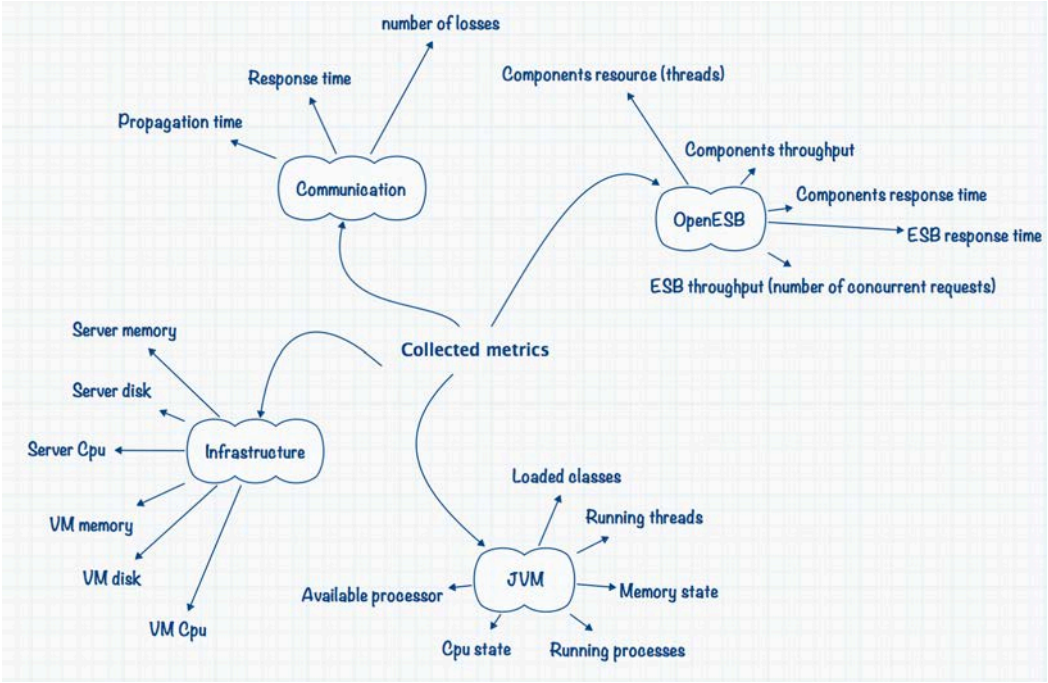


**Figure 4. 12 Summary of monitoring levels, parameters and metrics**

All the developed monitoring services are able to forward the collected metrics to the second part of the monitor component, which has to process the monitored data and recognize symptoms.

Next section introduces this second part of the Monitor component.

## 4.3 Symptoms detection module

The proposed ASB has the capability to identify critical situations and to react. This capability starts by the monitoring services presented in the previous section. These services collect data related to the state of the different levels that compose the ASB. Consequently, the collected data needs to be aggregated, filtered and correlated to identify the relationship between the different metrics and to detect symptoms. To support these symptoms detection features, a second step of the Monitor component based on event processing technologies is developed.

The symptom detection tasks based on event processing technologies ask for:

- a well definition of the structure of the monitoring data since it is shared by the monitoring services and the event-processing module;
- an identification of what can be a symptom;
- a definition of the processing rules that allows the detection symptom. These rules can be known at design time or learnt at runtime.

To cover these needs, we propose

- an understandable structure of monitoring data;
- two approaches to create a symptoms catalogue (*a set of "what can become wrong?"*) that guides the definition of the processing rules;
- a formal approach followed to define the processing rules.

In this section, the event processing technologies are introduced before presenting our approach to exploit them.

### 4.3.1 Event processing technologies

Event processing technologies can be used for tracking exceptional events or situations based on processing rules. The rules can be applied to an individual event (simple event processing) or to a large number of events (event stream processing and complex event processing). Event processing involves three kinds of actors:

- *event publisher* that generate the events stream;
- *event processing agent* that processes the events based on defined rules;
- *and event consumer* that receives result of the event processing agent.

The next figure 4.13 shows the inheritance hierarchy of event processing agent types [ETZ 10]. The figure 4.14 presents an example showing a connection of processing agents.

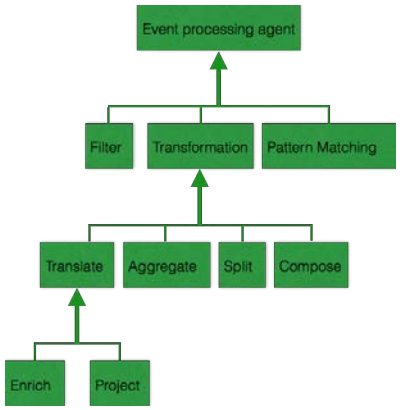


**Figure 4. 13 Event processing agent types**

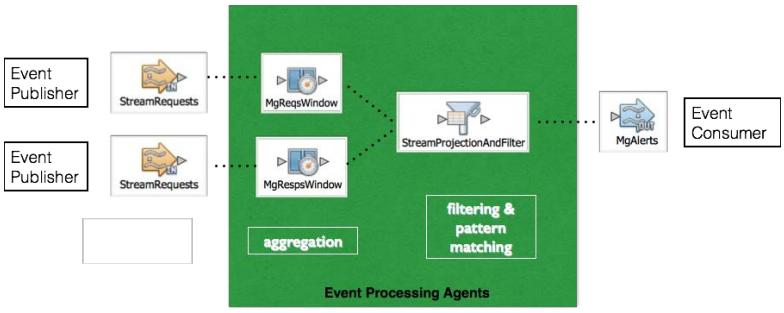


**Figure 4. 14 Illustration of event processing agents connection**

The event processing technology is used to process collected monitoring data and to detect symptoms that represent QoS or scalability degradations. Based on the design and the structure of the Monitor component (Figure 4.15) and the MAPE loop included in the proposed architecture of the ASB, we have:

- the multi-level monitoring services that act as the event producers;

- the Analyze component that acts as the event consumer;
- the complex event-processing module that has to detect symptoms from incoming events and to send them to the Analyze component.

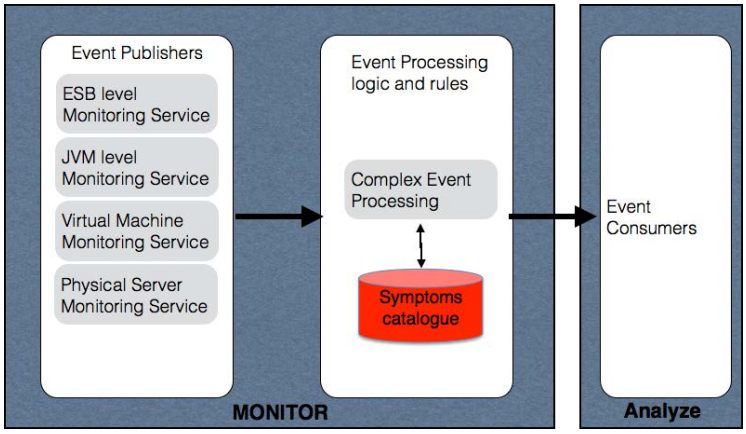


**Figure 4. 15 Event processing usage in the monitor component**

The following section details the two approaches to define the symptoms catalogue.

### 4.3.2 Symptoms definition approaches

The symptoms catalogue is a model that characterizes identified abnormal situations or critical states of monitored parameters. Two approaches have been proposed to define it:

- for the first approach, we correlate symptoms to the non-satisfaction of defined QoS requirements. To realize the approach, a model to allow the ESB administrator to impose QoS requirements is proposed. With this model, symptoms are correlated to a non-satisfaction of expressed requirements;
- for the second approach, we correlate symptoms to constraints and limits of the ESB and its deployment context regardless of QoS requirements defined by the ESB administrator. Approaches and methodologies to characterize ESB limits are studied to propose a solution that can be used to identify symptoms due to the ESB implementation and deployment context.

Hereafter, we detail these two proposed approaches.

*Semantic model for requirements expression*

The definition of symptoms can be done according to the requirements of the distributed systems defined by the ESB administrator. We propose a QoS-oriented semantic model to

allow expressing these requirements. The goal of this model is to characterize the QoS requirements and afterwards translate them in symptoms to be observed. These symptoms are detected if requirements are not satisfied.

Many approaches based on ontologies have been proposed for QoS requirements modelling. The most mentioned are DAML-QoS [ZHO 04], QoS-MO [TON 08], OWL-Q [KRI 07]. Following the same approach, we propose and implement an ontology. Including ESB solutions, the proposed ontology is applicable to the entire communication systems, which should ensure a certain level of QoS to users as it is based on the QoS management framework for communication systems proposed by the International Telecommunication Union (ITU) [ITU 97] and the UML profile for modelling QoS proposed by the Object Management Group (OMG) [OMG 08]. The figure 4.16 illustrates the concepts and the relationships of this ontology:

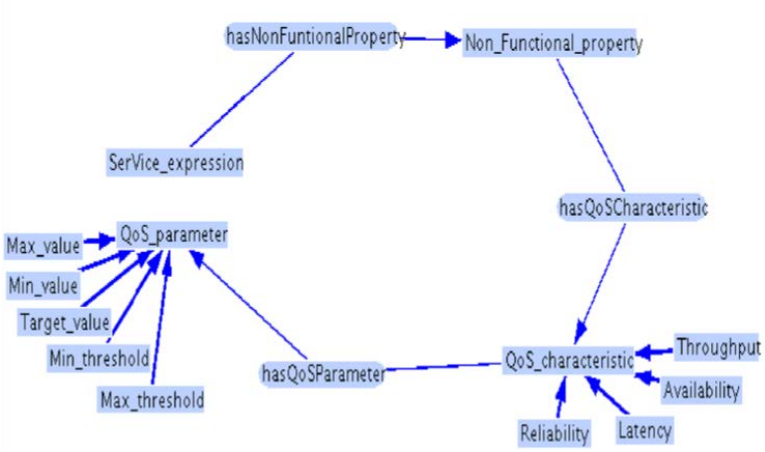


**Figure 4. 16 Ontology-based semantic expression**

- the *Service_expression* concept is one of the main classes of the ontology and represents the definition of a non-functional requirement. Each instance of this class allows to characterize a requirement that has the property (or attribute) *hasNonFunctionalProperty* to identify its non-functional properties;
- the *Non_functional_property* concept characterizes the non-functional properties of a *Service_expression*. This class has a single property *hasQoSCharacteristic* to identify the concerned QoS characteristics of the *Non_functional_property*;
- the *QoS_characteristic* concept makes possible the characterization of the quantifiable QoS properties (latency, throughput, availability, reliability, etc.) of a *Service_expression*. This class has a single property *hasQoSParameter* that defines the value of each characteristic involved in a *Service_expression*.

For each characteristic involved in the definition of a *Service_expression*, parameters are defined to express the accepted, desired and threshold values (i.e. minimum and maximum values). This information is used to define basic symptoms that can be observed to predict or to detect anomalies.

Next paragraphs present the second approach for the symptoms catalogue definition.

*ESB QoS model characterization for symptoms definition*

The definition of symptoms can be done by correlating them to constraints and limits of the ESB implementation and its deployment context. Indeed, the ESB can have QoS and scalability issues when it reaches limits and constraints due to the implementation or to the resources available for it [UEN 06]. Symptoms can be defined and correlated to these issues.

Several approaches can be followed to have a solution, platform or framework to define and characterize QoS and scalability issues due to limits and constraints of the implementation or the available resources. For instance:

- a real ESB implementation deployed in a production context can be used to observe the ESB's behaviour and to detect critical situations when they occur. However, in a real production context, the actions needed to study the ESB (e.g. message interruption, traffic generation) can have a real impact on the running business. Also when a critical situation occurs, it will be hard, even impossible, to replay it in order to better identify and learn what are limits of the ESB;
- an emulation solution can be used to simulate through a real ESB implementation the behaviour of service consumers and providers. The emulator allows playing and replaying scenarios that should correspond to the critical situations possible in a production context. However, the emulator deployment resources and scenarios need to be as much as possible similar to the production context;
- a simulation solution with models that simulate the behaviour of the ESB but also services consumers and providers can be used. Simulation tools are more and more exploited in distributed systems context, but results that can be got from them are really dependent on simulation models;
- queuing theory solutions with mathematical models that represent systems characteristics can also be used to have a more formal approach with analytics results.

These different approaches allow evaluating, identifying and characterizing QoS and scalability issues that may arise when an ESB is used. Based on the study of evaluation approaches and existing tools and solutions, the emulation solution can be used to simulate through a real ESB implementation the behaviour of service consumers and providers, and it allows playing and replaying scenarios that should correspond to the critical situations possible in a production context. Therefore, it has been chosen as the most efficient approach that can be followed to easily characterize ESB limits and constraints.

The proposed Emulation Platform for ESB Systems (EPES) presented in Chapter 3 allows easily simulating communications of service consumers and providers through a real ESB implementation to identify the stress points and qualify ESB limits. This information is used to define symptoms that can be observed to predict or to detect anomalies.

Next section presents an approach to formally define detection rules of well-known symptoms.

### 4.3.3 Formal approach for processing rules definition

By knowing the main characteristics of potential symptoms, the rules that can be implemented by any event processing technology solutions can be defined. In the literature, chronicles usage is one of existing formal approach that can be used to define these processing rules. “A chronicle is a set of observable events with some time constraints” [PEN 09]. An expert usually ensures the explicit definition of chronicles. Automatic learning solutions can also be used to discover new ones at runtime.

Following the chronicles definition approach, temporal evolutions of parameters associated to each undesirable states can be well characterized.

By considering basic symptoms that can be detected in the ASB context namely an exceeding of a threshold, a burst of values exceeding the threshold but oscillating, or a burst of values exceeding the threshold but following an increasing tendency (Figure 4.17), we define a set of chronicles.

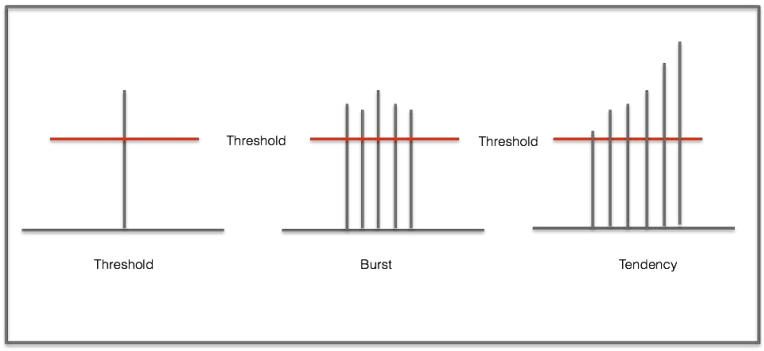


**Figure 4. 17 Basic symptoms illustration**

The equation 5 (eq 5) illustrates an example of chronicles usage in the ASB's context to define a rule that allows the detection of a tendency on the memory usage. This equation means that three events characterizing a percentage of memory usage that exceeds a defined threshold are observed in a short time interval.

$$Chronicle\ MemoryUsageTendancy$$

$$\{$$

$$Events\ \{(e1, t1), (e2, t2), (e3, t3)\}$$

$$Constraints\ \{(t2 - t1) < \varepsilon, (t3 - t2) < \varepsilon' and\ (\varepsilon - \varepsilon') \sim 0\ \}$$

$$\}$$

*(eq 5)*

*with $e_1$ an observed percentage of memory usage that exceeds defined threshold at $t_1$*

*with $e_2$ an observed percentage of memory usage that exceeds defined threshold at $t_2$*

*with $e_3$ an observed percentage of memory usage that exceeds defined threshold at $t_3$*

More complex rules can be defined based on the correlation of several levels monitoring parameters. An event processing language that depends on the technical event processing solution can be used to translate defined rules.

We also define a model to characterize the results of the event processing solution (Figure 4.18).

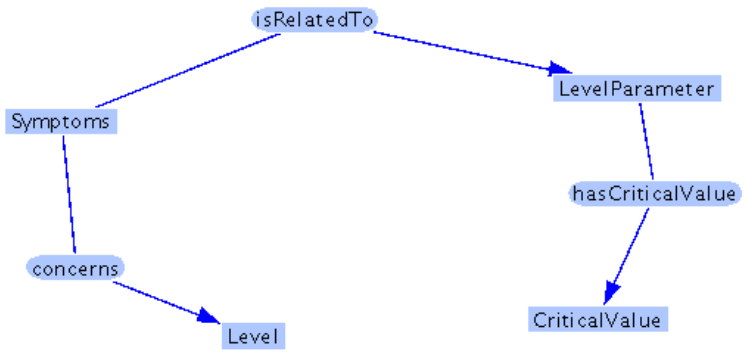


**Figure 4. 18 Semantic model for symptom characterization**

An instance of this model can be a symptom detected at the JVM level related to a very high and critical percentage usage of the memory that has a critical value at 95%. This characterization of the symptom has to be sent to the Analyze component that will make a diagnostic (a potential scalability issue due to the JVM memory for instance).

### 4.3.4 Summary

The complex event processing techniques have been exploited to satisfy the capability of the ASB to detect symptoms from collected data.

Approaches to identify symptoms to detect are proposed. They are based on the requirements of connected distributed systems or based on the limits of the ESB implementation and the deployment context. In this thesis, the proposed Emulation Platform for ESB Systems (EPES) presented in Chapter 3 is to well identify and characterize what can be the bottlenecks in an ESB configuration and deployment context.

Approach to formally define the rules to detect identified symptoms is also presented.

## 4.4 Conclusion

The Autonomic Service Bus aims at managing in a smart and autonomic way the scalability and the QoS offered to distributed systems during the integration and the mediation.

In this chapter, we have presented the proposed monitoring solution. This solution is firstly based on services developed to supervise all the layers that constitute the ASB (ESB instances

and underlying computing resources) and secondly on an event-processing module proposed for the treatment of the monitored data in order to recognize and detect critical situations.

Methodologies and approaches to define and characterize critical situations that may arise are proposed.

During the design and the development of the monitoring services, several approaches were studied to measure their cost and to choose the ones that allow proposing a lightweight monitoring solution in order to limit the impact on the ESB and the distributed systems operations.

An improvement and perspective of the approach is to adopt a solution to go towards a smart and self-adaptive monitoring solution. Indeed, the number of monitored parameters and the quantity of collected data can quickly increase. In this context, the monitoring may become an important limitation. Its activities can degrade the QoS and scalability of the ESB (for instance due to the resources consumed by monitoring services, or to the quantity of data to process). Either the observation of parameters can have a high cost, or the event-processing component that has to process collected data for symptoms detection can become the bottleneck. To better avoid that the monitoring itself becomes the limitation of the QoS and scalability management, an approach to go towards a dynamic and smart monitoring can be proposed. A way to control monitoring services for the data and metrics collection in order to make them self-adaptive will improve the scalability and the efficiency of the monitoring solution. The goal will be to reduce the cost of the monitoring and the quantity of data to be processed by the event-processing component by activating or deactivating dynamically part of the monitoring services in order to collect only data related to symptoms to detect or to predict.

At this managed maturity level (with monitoring solution), an administrator is always required to analyze the monitored data and initiate the adaptation plan to be executed. Next chapter presents solutions and models developed to guide and automatize the analysis of symptoms coming from the monitor component and the definition of the most adequate adaptation plan to be executed.

# Chapter 5. **Analysis and plan models**

**Contents**



## 5.1 Introduction

The architecture of the proposed Autonomic Service Bus results in the extension of existing ESB solutions with QoS and scalability oriented mechanisms and an autonomic manager that applies self-manageability functions.

The monitoring services of the autonomic manager have been designed and presented in Chapter 4. To go towards an autonomic maturity level, the autonomic manager has to ensure the analysis and the interpretation of the symptoms coming from the monitoring in order to define the adequate adaptation plan to be executed. The main goals of the autonomic manager's Analyze, Plan and Execute components are respectively [IBM 05]:

- to study and analyse symptoms received from the Monitor component and to diagnose potential anomalies that impose to apply changes on the managed system;
- to define and structure the set of actions to be executed when a request for change has been produced by the Analyze component in order to provide an adequate scalability and QoS level;
- to control, coordinate and orchestrate the execution of planned actions in order to deal with QoS and scalability degradations.

Several analysis approaches and solutions have been proposed:

- expert systems are generally used to produce a quick diagnostic based on the characteristics of well-known symptoms. For instance, the expert system can correlate their diagnostic to well-known evolutions of the system's characteristics and their frequencies (e.g. a high latency in holidays period means a high CPU load due to a high number of concurrent users). In [VIZ 13], a chronicle-based diagnostic approach is followed to define an expert system based on a set of chronicles. Their recognition (or detection) allows producing the diagnostic.
- more frequently a good diagnostic process requires a wide observation of the system to have its global state. And generally, statistical reasoning and probabilistic techniques are used to correlate several events, traces and logs in order to have as wide as possible a global view of the system's state before applying an analysis of detected symptoms [STE 05]. In [BIG 02], time series and Bayes classification solution are included to an agent for more complex analysis. In [STE 02], Bayesian Networks technologies are incorporating in the analysis process to identify the cause of issues. In [CAL 09], authors implement the analysis function by integrating the quantitative analysis tool PRISM [KWI 05] that can be used for the analysis of probabilistic models including discrete-and continuous-time Markov chains.

For the Analyze component proposed in this thesis, we follow the second approach and develop a probabilistic model that allows inferring the global state of the ASB based on received symptoms. Indeed, we exploit the proposed Emulation platform for ESB System (presented in Chapter 3) to build a model that contains the dependencies between several performance indicators that can be retrieved from the developed monitoring services (e.g. end-to-end response time, latency through the bus, loss rate, ESB load in terms of number of concurrent messages through the bus, java virtual machine load in terms of heap memory, CPU usage and running threads, virtual machine and physical server load in terms of memory and CPU usage, etc.). This model helps to make the diagnostic and to have a global view of all of the parameters of the ASB based on the state and value of part of them represented by the symptoms.

Regarding the plan functionalities, they range from simple adaptation processes to a complex ones and most of existing approaches to guide the definition of these adaptation processes are based on policies or architectural models [MAR 08]:

- policies used to guide the decision process are expressed using [KEP 03] [IBM 05]:
    - event-condition-action (ECA) policies based on "if-then-else" rules that guide the process of selecting the adequate actions to apply on the system according to its state. The difficulties with this approach are the definition of the rules by an expert, their update to keep them as most as possible exhaustive and the management of conflicts that can occur between them;
    - goal-based policies that describe and characterize the desirable states that guide the process of selecting the adequate actions to apply to be in these states. The difficulties with this approach are the well definition of desirable and undesirable states and a well characterization of the possible actions;
    - utility function policies based on an objective function that guides the process of selecting the adequate actions to apply to be in a desirable state. This approach improves the goal-based approach by providing desirability values that differentiate the possible states of the system. For instance, when there is no solution to identify the most desirable state, the best undesirable state can be achieved. The difficulties with this approach are the way to build the function and the different coefficients;
- architectural-model based approach can also be followed to guide the definition of the adaptation processes. Models are used to represent the system's behaviour, its requirements and the possible adaption plans. This allows applying the good adaptation plan that is consistent with the expected state model [MAR 08]. For instance, in [DAS 02], models are defined based on monitored data (an old and a new model), and the adaptation plan is defined according to the difference between the two models. In [GAR 02], models are used to well describe components and connectors of the system. A language is used to characterize repair strategies and how the solutions may be composed when adaptation is needed. Formal methods based on models of the self-adaptive system and models of properties to verify whether the system satisfied its goals are followed in [IFT 12] and [IGL 13] to provide evidence that defined plans allow achieving the system goals.

In this thesis, we develop a model-based approach based a formal characterization of the proposed QoS and scalability-oriented mechanisms for the Plan component. This model helps to define the best mechanism or composition of mechanisms that must be used in front of detected symptoms. Examples of adaptation processes to be executed according to the

selected mechanisms are also provided. To develop this model, we also exploit the proposed Emulation platform for ESB System to evaluate and characterize well the proposed mechanisms by identifying the symptoms they can correct and the costs of their usage.

The execution functions are generally based on services that implement the adaptation processes defined by the Plan component. In the ASB context, several services are developed to control and coordinate the execution of mechanisms proposed to correct QoS and scalability degradation. These services allow integrating the mechanisms to the ESB and the computing resources during the mediation processes to take into account requirements and constraints of distributed systems. They can also be used to modify the behaviour of already integrated mechanisms.

This chapter that shows more how existing approaches can be applied in the ASB context in order to have a closed MAPE loop is organized in three sections: section 5.2 presents the solutions to analyse symptoms, produce a diagnostic and identify if a change is required. Section 5.3 presents a way to correlate a specific symptom to a corrective solution during the Plan phase and introduces the execution services. Section 5.4 concludes the chapter.

## 5.2 ASB Analyze component

By following the autonomic computing incremental approach [IBM 05], adding the analysis functionalities to what we have presented in the managed maturity level (Chapter 4, Figure 4.1) helps to make the system predictive (a maturity level before autonomic). The role of an ASB administrator is here to initiate the good actions to execute according to the request for change (Figure 5.1).

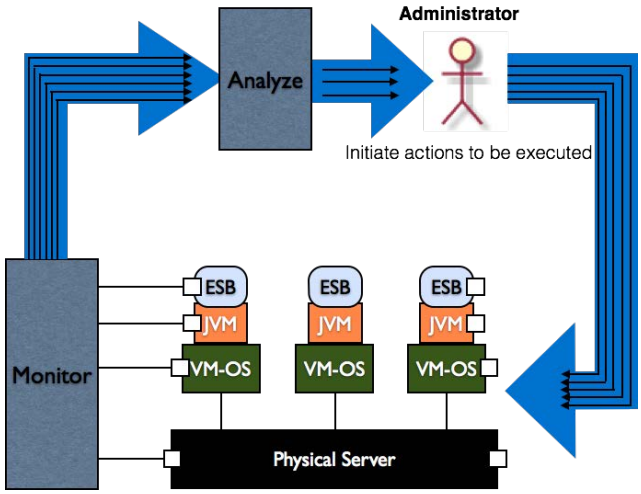


**Figure 5. 1 Predictive maturity level**

The main goals of the autonomic manager's Analyze component are to process symptoms received from the Monitor component and to produce a diagnostic. The output of the component is a request for change that guides the definition of the adaptation processes. Next section introduces its structural design.

### 5.2.1 Analyze component structural design

The figure 5.2 gives the proposed structure of the ASB's Analyze component divided in two steps:

- a step 1 for the analysis of symptoms received from the Monitor component based on a *Diagnostic service* that implements the needed allowing predicting or detecting degradation, but also identifying their causes if needed. The service uses a *Diagnostic model* (database in red) that belongs to the knowledge base.
- a step 2 based on a *Rules based system* integrated to define the criticality of the diagnostic results and the appropriate request for change to be sent to the Plan component.

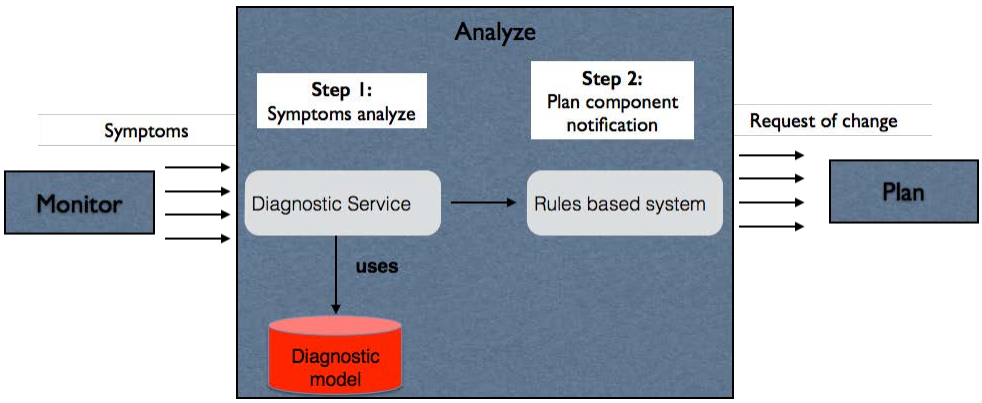


**Figure 5. 2 Analyze component structure**

Next sections detail these two steps. We present how the probabilistic approach has been followed to build the diagnostic model. Illustration of this model has been made thanks to EPES. The rules based system that constitutes the second step is also presented.

### 5.2.2 Probabilistic reasoning model for a smart diagnostic

Several approaches can be followed to develop the diagnostic solutions for symptoms analysis. Expert systems are generally used to produce a quick diagnostic based on the characteristics of well-known symptoms. However, Expert systems present common limits

such as the need of a high expertise, the difficulty to have an exhaustive list of cases, the need to be regularly updated, the management of their consistency, etc.

Moreover, more frequently a good diagnostic process requires a wide observation of the system to have its global state. For instance, in the case of medical diagnostic, this can be assimilated to the additional tests (e.g. sanguine, radiological, etc.), which are needed to determine the disease and to help to propose a treatment. In general this is needed if few symptoms have similar characteristics or if disparate symptoms and causes identify the same diseases. Similar situations are highly present in the ASB's context. For instance, a symptom related to a high propagation time can depend on several situations (e.g. overload of ESB components or overload of the service provider). Therefore, if this kind of symptom is detected, having a global view of the system state will allow making a good diagnostic.

Therefore in contrast to expert systems, we develop a probabilistic model for the Analyze component that allows inferring the global state of the ASB based on received symptoms. Several statistical models, probabilistic techniques, artificial intelligence principles (e.g. pattern recognition, case based reasoning, hybrid survivability models, Bayesian Networks (BN) and its derivations like as Naive Bayes and Markov Networks, etc.) can be followed to implement the diagnostic model required for the ASB.

The adopted approach in this thesis is based on a Bayesian Networks (BN) constituted by all the variables and parameters that the monitoring services can collect and by the conditional dependencies between these parameters.

Human experts may define the structure of this BN. However this task may be tedious when many variables are involved, or when the BN needs to be taken updated regularly. In this context, the proposed solution includes a structure-learning algorithm based on the PC (Peter-Clark) algorithm [SPI 01] that allows the discovery of the BN structure and its update. The BN establishes relationships between all the ASB's context variables and parameters. Probabilistic reasoning can be applied on the BN to make inferences about the overall state of the ASB using only the value or the states of variables represented by the symptoms.

Hereafter, concepts related to the Bayesian Networks (BN), the probabilistic reasoning and the PC algorithm are introduced.

*Foundation concepts introduction*

**Bayesian Networks.** A BN is a probabilistic graphical model that consists in a set of variables, each one with mutually exclusive states. Suppose we have a set of variables V={$v_1$, $v_2$, … , $v_n$} with a global probability distribution P; A and B two subsets of variables in V; the basic principle of BN is the conditional probability denoted by P (A = a/B = b) which means "the probability of A being in state a under the constraint that B is in state b" [BAR 12]. The conditional probability is defined by the Bayes' theorem (Equation 1 – eq 1 and Equation 2 – eq 2).

$$P(A|\mathrm{B}) = P(A \cap B)/\mathrm{P(B)}\ (eq\ 1)$$

$$P(A = a|\mathrm{B} = \mathrm{b}) = P(A = a\ \cap B = b)/P(B = b)\ \ (eq\ 2)$$

When the sets of variables A and B are conditionally independent, then P (A|B) = P (A).

When we have another variable C with P (A, B/C) = P (A/C) P (B/C), then the sets of variables A and B are independent given the set C.

The BN gives a Directed Acyclic Graph (DAG) that represents conditional dependencies (directed edges) among the variables (nodes) and conditional probabilities tables (CPT). The DAG can be used to represent both causal hypothesis and a set of probability distributions [SPI 95]. Once the graph is established, reasoning can be applied to infer the states of the variables.

The figure 5.3 shows a BN with the variable set V={A, B, C, D} and the global probability distribution P (A, B, C, D) = P (D|C) P (C|A, B) P (A) P (B).

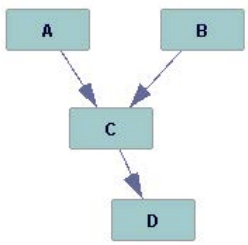


**Figure 5. 3 Example of Bayesian network**

**Probabilistic Reasoning.** When the relevant contextual variables are identified, a probabilistic model can be developed to represent the relations among these variables. The probabilistic reasoning consists in the capability to get how the evidence of some variables in an observed state affects the others. The reasoning is performed using the Bayes' theorem and the edges in the DAG to calculate the posterior probabilistic for each state of the variables, which represent the effects of the evidence [BAR 12]. The equation 3 (eq 3) shows how to calculate the most probable state (θ*) of the variable $v_1$ given the evidence E in the variable $v_2$.

$$\theta^* = argmax\,(\theta) * p\,(v_1 = \theta | v_2 = E) \qquad (eq\ 3)$$

In the ASB diagnostic model, the evidence is represented by detected symptoms, which are for us critical states in some variables. The reasoning result gives the diagnostic result.

**PC algorithm.** A BN allows capturing knowledge about a domain. However, techniques are needed to learn the BN's structure from the stochastic properties of the domain. A runtime learning and a update process are also important. They help to improve the precision of the inferences since the knowledge represented by the BN could change over the time and the relationships among the variables and the conditional probabilities tables could be different under different conditions. The PC algorithm presented in [SPI 01] gives an approach to learn the BN's structure and the DAG associated to it. The approach consists in performing tests for dependency in a data sample to discover the relationships between the variables [DEV 06].

*Design of the diagnostic model solution*

Three main modules have been proposed to build the probabilistic diagnostic model based on the introduced foundation concepts namely: a *Data process module*, a *Learn module* and a *Query module* (Figure 5.4). Following paragraphs detail their features.

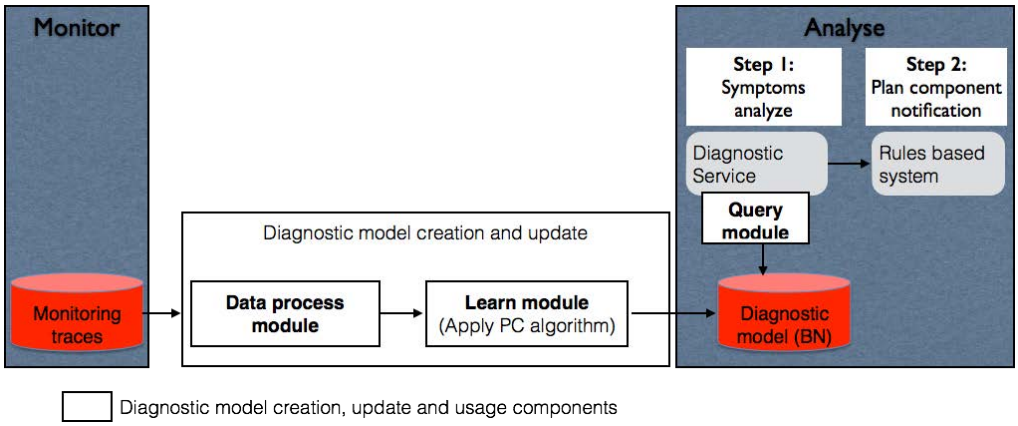


**Figure 5. 4 Probabilistic diagnostic model management structure and process**

**Data process module.** The monitoring can observe continuous or discrete parameters from the different level (ESB level and computing resources level). To reduce complexity of data processing, the values of all the observed parameters are translated to discrete values. The discrete values represent the state of the parameters in the BN.

Each parameter has defined states represented by intervals. By interval matching, the data process module translates the continuous values to discrete values and classifies all data sent from the Monitor component to the defined states.

For instance, the value v for the variable A corresponds to the variable A in the state a, if the interval of the state "a" includes the value "v" (Equation 4 – eq 4).

$$value\ (A) = v\ and\ v \in [a_i]\ \ (set\ of\ value\ of\ the\ state\ a)\ \Rightarrow A\ is\ in\ the\ state\ a$$

*(eq 4)*

The output of the Data process module is a data set with the different states of all the parameters. The Learn module uses this data set to discover the structure of BN.

**Learn module.** The Learn module is responsible of building the initial BN and keeping it updated during the execution time. It applies the PC algorithm on the data set coming from the Data process module to discover the BN and the dependencies between the variables. The resulting BN is a new knowledge base that offers a current state of the parameters.

Through the Query module, the diagnostic service can use this knowledge base to infer the global state of the system. Counting the occurrence of cases in the data set and applying the principle of the conditional probability allow making this inference.

**Query module.** The Query module is used by the diagnostic service to extract knowledge from the BN and to have the state of the whole system. The diagnostic service inserts received symptoms in the BN as evidences that only contain information about some parameters, and from this, it is possible to know about the remaining parameters in the BN.

The proposed diagnostic solution based on probabilistic dependencies is efficient as it allows getting a global state of the ASB based only symptoms received from the Monitor component. For instance, if making a diagnostic requires knowing the state of several parameters (e.g. x, y, z and t) and symptoms received from the Monitor component only give information on part of the parameters (e.g. x and y), then the model can be used to infer the state of the others (e.g. z and t).

Next section illustrates how EPES can be used to construct the model.

### 5.2.3 EPES usage for the construction of the diagnostic model

The model can be built at design time and updated periodically based on runtime monitoring data and the proposed Emulation Platform for ESB System (presented in Chapter 3) can be an efficient tool for that. The EPES platform can be used to collect data samples from a set of scenarios characterizing any kinds of businesses. These data samples are used to construct the probabilistic model. Illustration and validation of EPES usage for the construction of the diagnostic model are presented hereafter.

Several scenarios were simulated using EPES and the following topology (Figure 5.5). The scenarios configurations have been defined in the table 5.1. We use several configurations (Table 5.1) to have different workloads through the ESB. The number of requests has been set to values comprised between 20 and 1900. The requests are sent in a bust mode and the provider takes one second before sending a response. The requests and responses sizes are set up to 10 bytes.

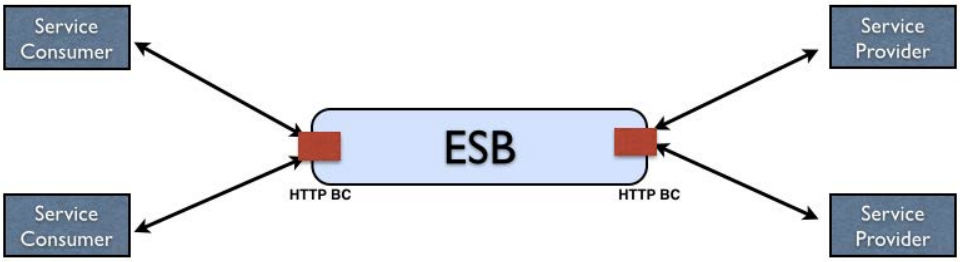


**Figure 5. 5 Simulation topology**

| Scenarios | Configurations | | | | |
|---|---|---|---|---|---|
| | Consumers side | | | Providers side | |
| | Number of requests | Sending mode | Requests size (bytes) | Processing time (s) | Responses size (bytes) |
| **1** | 20 | Burst | 10 | 1 | 10 |
| **2** | 100 | Burst | 10 | 1 | 10 |
| **3** | 200 | Burst | 10 | 1 | 10 |
| **4** | 400 | Burst | 10 | 1 | 10 |
| **5** | 800 | Burst | 10 | 1 | 10 |
| **6** | 1000 | Burst | 10 | 1 | 10 |
| **7** | 1200 | Burst | 10 | 1 | 10 |
| **8** | 1500 | Burst | 10 | 1 | 10 |
| **9** | 1600 | Burst | 10 | 1 | 10 |
| **10** | 1800 | Burst | 10 | 1 | 10 |
| **11** | 1900 | Burst | 10 | 1 | 10 |

**Table 5. 1 Scenarios configuration**

For each request mediated by the ESB, we collected data related to the state of the ESB and the underlying computing resources, for instance the CPU Load (CPU), the Heap Memory Used (MEM), the Thread usage percentage (TH), the propagation time through the ESB (PT) and the end-to-end Response Time (RT) (*from the service consumer point of view by taking into account the service provider processing time*). We include also the number of requests (REQ) sent in each scenario to correlate the state of the variables with the conditions under which the ESB is working.

After running the scenarios and collecting a large data set, performing the data pre-processing consists in defining the states and intervals for each monitored variables. This allows discretizing the values for a more fast learning of the Bayesian Network (BN) structure. The table 5.2 shows these variables and the defined states. The definition of the states and intervals depends on the scenarios and the configuration for the number of the requests, and on the recommendations of the ITU G.1010 for the propagation and the response time.

| Variables / Parameters | States and intervals |
|---|---|
| CPU (CPU) (percentage) | Low=[0, 25); Medium=[25, 50); High=[50,75); Very High=[75,100] |
| Memory (MEM) (percentage) | Low=[0, 25); Medium=[25, 50); High=[50,75); Very High=[75,100] |
| Thread (TH) (thread usage percentage) | Low=[0, 25); Medium=[25, 50); High=[50,75); Very High=[75,100] |
| Propagation time (PT) (in seconds) | Preferable=[0, 1); Acceptable=[1, 2); Unacceptable=[2, +∞) |
| Response time (RT) (in seconds) | Preferable=[0, 2); Acceptable=[2, 4); Unacceptable=[4, +∞) |
| Number of requests (REQ) | Low=[0, 1000); Medium=[1000, 2000); High=[2000,3000]; |

**Table 5. 2 Monitoring parameters states and intervals**

*Bayesian network construction*

Once the data pre-processing done, the algorithm can be applied to build an initial model that represents the state of the ESB in different contexts. We used TETRAD IV version 4.3.10-6, which allows executing the PC algorithm to discover the directed acyclic graph that represents the relations of the variables, and calculates the conditional probabilities table that defines the BN.

The figure 5.6 shows the obtained BN. This BN shows relations between the different characteristics, however let us recall that the edges in the resulting BN do not represent causal relations; instead those are useful to define independence properties. Another BN built with another dataset and different workload conditions is available in [DZU 13].

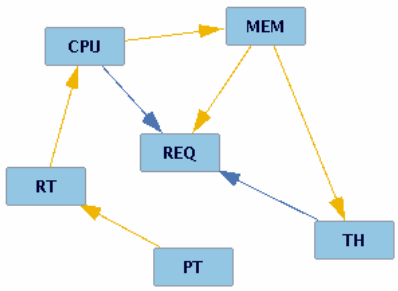


**Figure 5. 6 Discovered Bayesian network**

*Example of query results*

When a state is observed in some variables (evidences), the independence relations allow knowing how other variables are affected by means of the probabilistic reasoning. For example, the most probable state of the CPU in the BN generated is P (CPU = VH) = 0,8597. When we insert this evidence in the BN, the posterior probabilities allow us to infer and know what is the most probable state of each other variables (Table 5.3).

| Variables | States |
|---|---|
| **MEM** | P(MEM=H \| CPU=VH) = 0,5430 |
| **TH** | P(TH=VH \| CPU=VH) = 0,9935 |
| **REQ** | P(REQ=H \| CPU=VH) = 0,4973 |
| **PT** | P(PT=U \| CPU=VH) = 0,9898 |
| **RT** | P(RT=U \| CPU=VH) = 0,9616 |

**Table 5. 3 Query results and parameters states inference**

These values calculated using TEDRAD IV mean:

- if the CPU is VH, then the probability to have the propagation time and the response time in an unacceptable state is respectively 0,98 and 0,96;
- if the CPU is VH, then the probability to have a high usage of the memory is 0.54;
- if the CPU is VH, then the probability to have a high number of concurrent request is 0.49;
- if the CPU is VH, then the probability to have a high percentage of ESB thread usage is 0.99.

So by only receiving symptoms related to the CPU, other parameters states can be inferred to make a diagnostic. This inference is useful to define the rules for the definition of the requests of change. Illustration is presented in next section.

### 5.2.4 Rules based system for request for change definition

The introduced rules based system that constitutes the second step of the proposed Analyze component helps to define the criticality of the diagnostic results and the appropriate request for change to be sent to the Plan component for the definition of the adaptation process.

According to the diagnostic results, several “if-then-else” rules can be defined to classify diagnostic results based on the criticality of issues to correct (*WARNING*, *CRITICAL*, *ERROR*). Accordingly, a request for change can be formulated to express a need of solution for prevention or for correction.

For instance, next equation 5 (eq 5) gives an example of request for change that can be sent to the Plan component based on the diagnostic results (query results in the previous section). This request for change describes well the detected symptom (critical CPU), the impact on the propagation time and the response time and the states of the other parameters to help the adaptation process definition.

$$RequestForChange\ DecreaseCPUusage\ \{$$

$$DetectedSymptom\ \{Critical\ CPU\ level\}$$

$$Potential\ impact\ \{Undesirable\ propagation\ time, undesirable\ response\ time\}$$

$$System\ State\ \{Memory\ correct, number\ of\ concurrent\ request\ correct\}$$

$$\}$$

*(eq 5)*

This request will be exploited by the Plan component, which is responsible to determine, the required actions to recover the normal operational level of the CPU (e.g. add more CPU if the physical server can support it, increase the number of threads that can be used by the ESB components since the state of the memory is not critical, reduce the number of incoming requests, or apply another plan).

### 5.2.5 Summary

The Analyze component is proposed to process symptoms received from the Monitor component and to produce a diagnostic. The output of the component is a request for change (sent to the Plan component in a complete autonomic manager MAPE loop).

We develop a probabilistic diagnostic model that offers solutions to analyse symptoms received from the Monitor component. The model needs to be well trained and solution similar to the EPES platform helps for that. Rules based system is introduced to define the type of request for change that needs to be sent to guide the treatment plan.

The proposed probabilistic diagnostic model can be improved by covering a big challenge that consist in defining the symptoms (to be observed) that allows inferring the state of the whole ASB. Indeed, it is needed to have a solution that helps to identify the critical parameters and variables that need to be observed according to their monitoring cost but also according to their usability to infer the other parameters.

Next section presents approaches and models followed for the Plan and Execution components.

## 5.3 ASB Plan and Execute components

The main goal of the Plan component of the autonomic manager control loop is to run a decision process that defines the actions to be executed after a specific diagnostic in order to satisfy QoS and scalability requirements. The Execute component for its part provides functionalities to perform changes on a managed system by modifying partially or totally its structure or behaviour.

By adding the Plan and Execute components, the whole control loop is covered and the role of an ASB administrator is here to define the semantics models and the inference rules based on self-management goals (Figure 5.7). These functionalities help to make the system adaptive and to go towards the autonomic level.

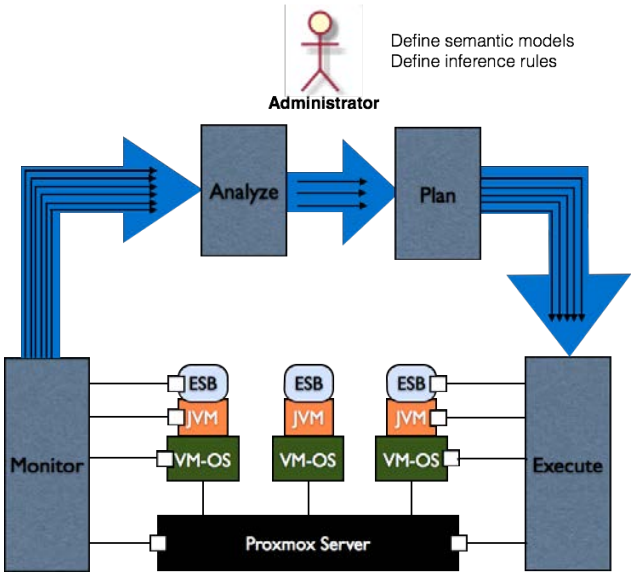


**Figure 5. 7 Adaptive maturity level**

### 5.3.1 Plan and Execute components structural design

The figure 5.8 gives the proposed structure of the ASB's Plan component based on two steps:

- a step 1 based on a *Plan definition Service* that implements the needed solutions to define the adaptation processes. Models (database in red) that belong to the knowledge base are used to characterize the goals and the adaptation strategies;
- a step 2 based on a *Plan transfer Service* that sends the change plan to the execute component.

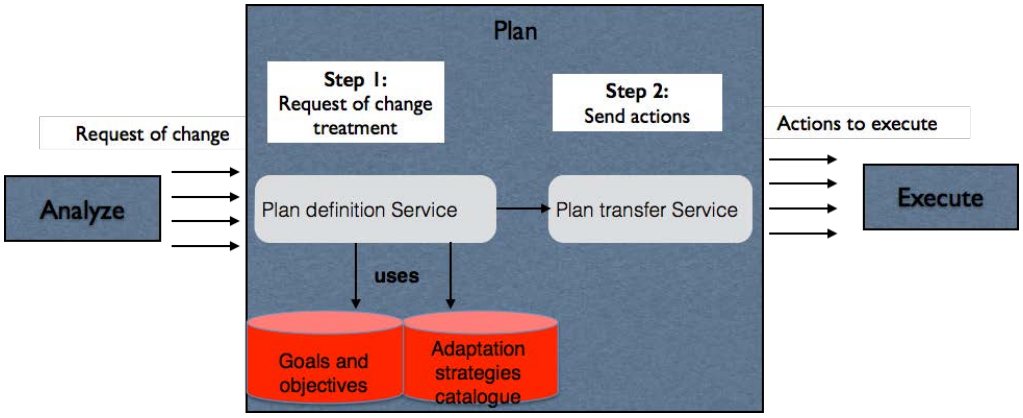


**Figure 5. 8 Plan component structure**

The proposed Execute component is also composed by two steps (Figure 5.9):

- the first step receives the execution requests. Its main role is to identify, orchestrate and coordinate services able to execute the actions on specific components and mechanisms (based on the targeted levels);
- the second step is composed of services that apply the requests on the specific components and mechanisms.

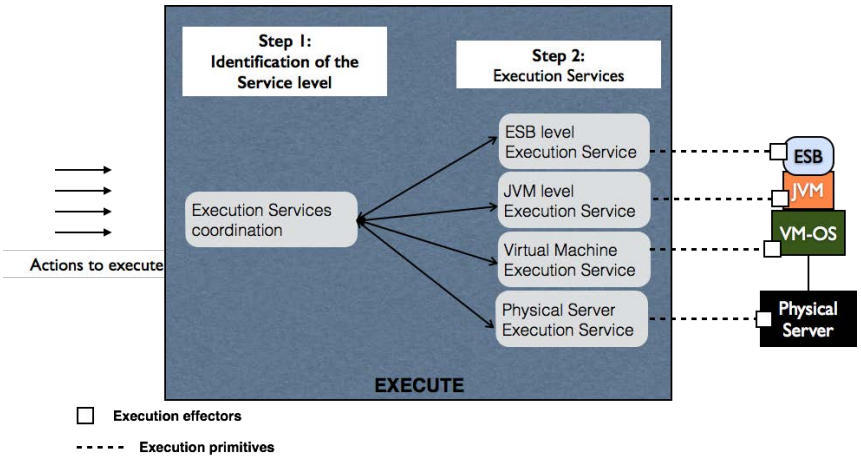


**Figure 5. 9 Execute component structure**

Next sections detail these two components. We present how the model-based approach can be followed to define the plan based on possible actions that can be supported by the ASB. A formal characterization of the proposed QoS and scalability-oriented mechanisms (e.g. intra-bus and extra-bus mechanisms detailed in the chapter 3) to define the best mechanism or composition of mechanisms that must be used in front of the detected symptoms is developed. Illustration has been made to show how EPES can be used to develop this characterization. Examples of adaptation processes to be executed according to the selected mechanisms are also defined before presenting the execution services.

### 5.3.2 Model driven approach for a plan model

An important part of execution actions consists in proposed mechanisms to be enforced in order to provide an adequate scalability and QoS to the integration. These mechanisms can be more efficient and better suited in specific situations than others. For instance, when a service or a bus component is saturated, a flow control mechanism can be implemented. It has a positive effect on data transfer reliability; but it has also a cost expressed in terms of additional delay because the emission of the requests is deferred.

An ontology-based semantic model can be used to choose the best mechanism or composition of mechanisms able to satisfy the expected scalability and QoS to provide a formalized classification of the mechanisms we consider for the ASB and to choose the best mechanism or composition of mechanisms able to satisfy the expected scalability and QoS.

The proposed model and an inferring approach aimed at identifying and choosing the adequate mechanisms composition and their parameterization are presented hereafter.

The proposed ontology (Figure 5.10) illustrates the different concepts and their relationships. Here, we briefly explain its main concepts and attributes:

- the concept *Mechanisms* is the main concept. Each instance of this class allows characterizing one of the intra-bus or extra-bus mechanisms that are involved. A mechanism has several properties (or attributes);
- the concept *ControlCommand* gives the definition of the effectors interface to use for achieving the mechanism;
- the concept *Level*s characterizes the target level of the mechanism namely at the intra-bus level (ESB components and processes) or at the extra-bus level (JVM, virtual and physical IT resources);
- the concept *Goals* allows characterizing the non-functional goals of a mechanism. This concept has the following properties. For instance, “Congestion Aware” and “Delay Aware” could be possible instances of this concept;
- the concept *Costs* describes the cost of using a certain mechanism. For example, the cost of using the selective discarding mechanism is quality degradation;

− the concept *ConfigurableParameters* characterizes the configurable parameters of a mechanism. For instance, a selective discarding mechanism may be parameterized through rate of requests to discard.

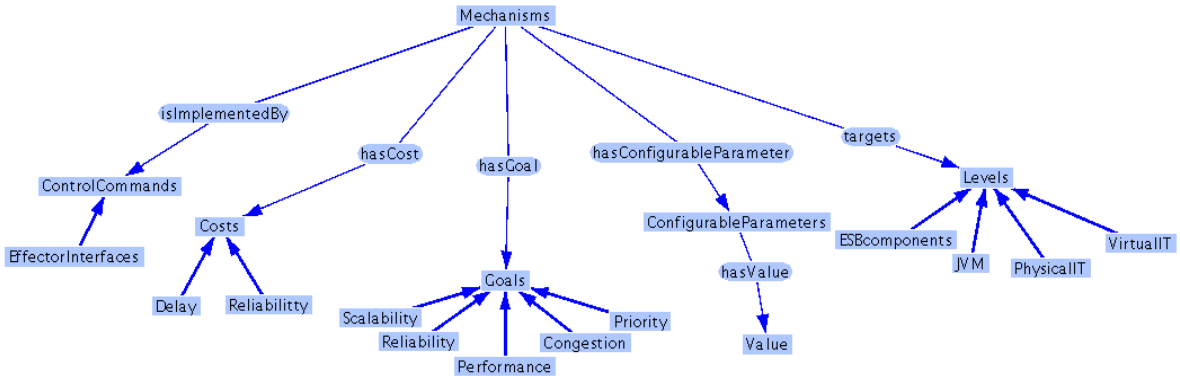


**Figure 5. 10 Ontology model for QoS-based mechanism classification**

The ontology-based model has been developed in order to define a set of concepts and a vocabulary to characterize semantically the intra-bus and extra-bus mechanisms. Such characterization allows producing the necessary knowledge to the plan definition for a dynamic choice, configuration and deployment of the appropriate corrective mechanisms according to a given context.

Next section illustrates how EPES can be used to instantiate the model.

### 5.3.3 EPES usage for intra-bus and extra-bus mechanisms characterization

To instantiate the ontology model for mechanisms characterization, the EPES platform is used to test and validate the implemented intra-bus and extra-bus mechanisms for QoS and scalability management.

For instance, one of the mechanisms that can be applied when anomalies occur is to tune the number of threads that can be used by the ESB components. Indeed, the ESB binding components and service engines have a set of configurable parameters; one interesting parameter is the number of threads that can be allocated to the components. These threads are used by the component to perform their tasks. This characteristic is exploited to propose a mechanism that allows tuning dynamically the number of allocated threads. However, before using this mechanism in a specific context, it is good to know how it can improve the QoS and scalability requirements and what should be its impacts.

For that, we use EPES to show that when a binding component does not have enough threads, it will not accept incoming requests and losses will occur. We later present the impact of tuning these threads configuration.

We use the HTTP binding component (BC) and put a first configuration (*Config 1*) with 10 threads and a second configuration (*Config 2*) with 100 threads. Several scenarios were simulated using EPES and the same topology on the figure 5.5. The scenarios configurations are similar to the ones defined on the table 5.1, but the number of requests has been set to values comprised between 40 and 3800.

Figure 5.11 and table 6.4 show the detail of obtained results:

- the curve labelled *Config 1* gives the average propagation time for scenario supported by the first configuration without losses. Only the height first scenarios were supported (3000 requests) when the http BC was configured with 10 threads;
- the curve labelled *Config 2* gives the average propagation time for scenario supported by the second configuration without losses. With this configuration (100 threads) the ESB is able to support more than 3000 requests.

The conclusion of figure 5.11 is that the mechanism that allows tuning dynamically at runtime the threads allocated to the BC has an impact on the reliability. More incoming requests are accepted. However we can notice that the mechanism has an impact on the propagation time through the ESB that increases more quickly between 3000 requests to 3800 requests compared to 40 to 3000 requests. The figure 5.12 shows how the gradient of the propagation time (in degrees) increases between 0 and 3000 requests and between 3000 and 3800 requests. These results can be explained by the fact that there are more incoming requests that are accepted by the ESB when the number of thread increases. These requests will spend more time waiting through the ESB for the rest of the mediation process.

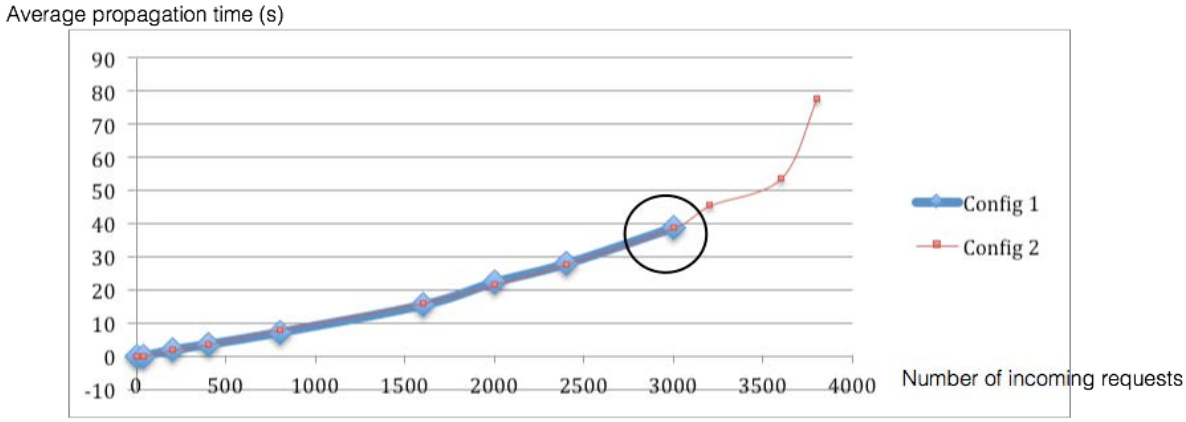


**Figure 5. 11 Impact of the HTTP BC configuration tuning**

| Scenarios | Configuration | Performance results | | | |
|---|---|---|---|---|---|
| | | Config 1 HTTP BC | | Config 2 HTTP BC | |
| | Total number of incoming requests | Losses | Average Propagation time (in s) | Losses | Average Propagation time (in s) |
| **1** | 40 | NO | 0,03 | NO | 0,03 |
| **2** | 200 | NO | 1,91 | NO | 1,90 |
| **3** | 400 | NO | 3,56 | NO | 3,58 |
| **4** | 800 | NO | 7,20 | NO | 8,03 |
| **5** | 1600 | NO | 15,46 | NO | 16,16 |
| **6** | 2000 | NO | 22,30 | NO | 21,66 |
| **7** | 2400 | NO | 27,93 | NO | 27,83 |
| **8** | 3000 | NO | 38,83 | NO | 38,82 |
| **9** | 3200 | YES | - | NO | 45,52 |
| **10** | 3600 | YES | - | NO | 53,83 |
| **11** | 3800 | YES | - | NO | 77,96 |

**Table 5. 4 Results of the HTTP BC tuning**

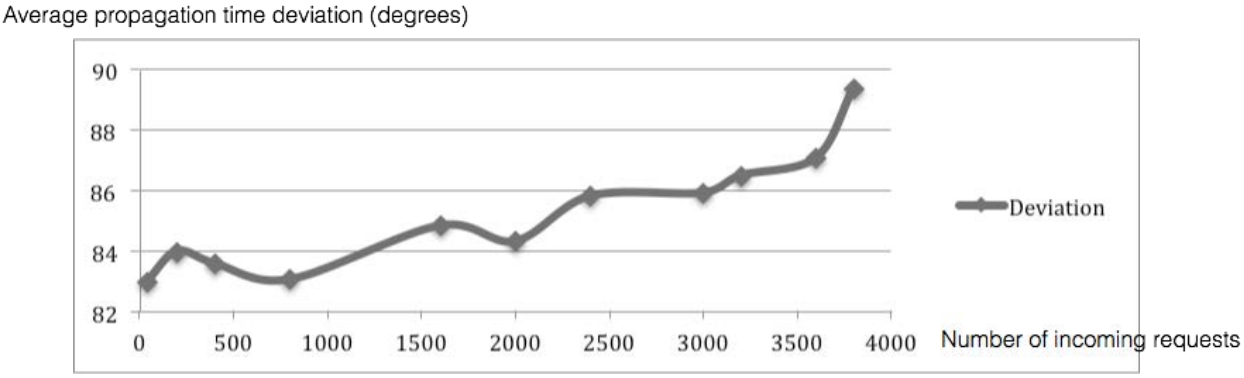


**Figure 5. 12 Evolution of the propagation time deviation**

Based on these results, we can define the cost and advantage of the mechanism to instantiate the ontology model for mechanisms characterization (Figure 5.13). The *tuningBCThread* mechanism targets the *ESBcomponents* level. The *numberOfallocatedThread* is its configurable parameter. This mechanism has goal as it leads to improve the *reliability* by reducing losses at the entry point of the ESB. However, using this mechanism has a cost expressed in terms of *delay*. The *ESBlevelExecutionService* controls the execution of this mechanism.

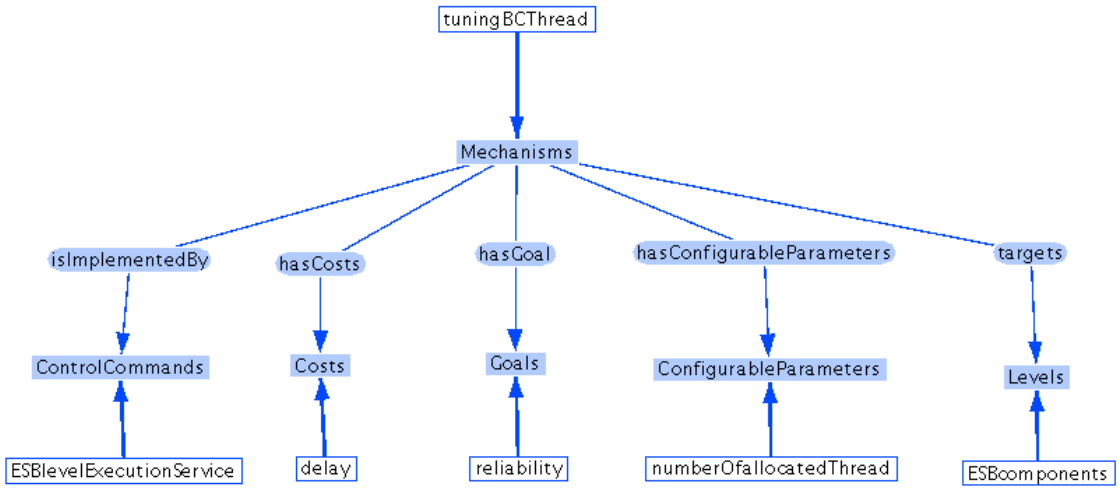


**Figure 5. 13 Plan ontology instance**

Next section provides examples of adaptation processes to be executed according to the selected mechanisms.

### 5.3.4 Adaptation processes examples

When relevant symptoms are detected, the analysis function raises a request for change based on the fact that the observed state does not fit the expected state. The request is sent to the Plan component and the instantiated ontology model is queried. The ontology reasoner deals with the query by searching among the characterized mechanisms instances the one to use.

According to the chosen mechanisms and their design an adaptation process is defined to identify the sequence of actions that need to be executed to perform the mechanisms.

The adaptation process can be simple based on the use of a simple mechanism that implements all the needed steps to achieve its goal by involving a few components (generally one). In that case, the adaptation process is limited to the activation or deactivation of the mechanisms with the possibility to configure and tune their behaviour (Figure 5.14). For instance the selective discarding mechanism to implement the partial reliability can be implemented by the binding component at the consumer side. An adaptation process based on this mechanism consists in tuning the configuration of the binding component to integrate in its behaviour the selective discarding. Parameter of the mechanism such as the percentage of requests to discard can be configured.

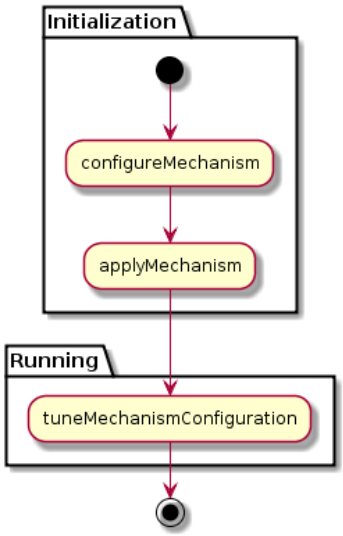


**Figure 5. 14 Simple plan process**

The adaptation process can be complex based on the use of a complex mechanism that involves more than one component and that required several actions to be executed in sequence or in parallel. For instance, the increase of the cluster's instances to implement the horizontal elasticity concept requires several actions to be implemented: deployment of a new instance, add of the new instance to the cluster, configuration of the load balancer to take into account the deployed instance (Figure 5.15). A process based on the orchestration of these actions needs to be executed.

The adaptation process can be even more complex based on the coordination of several processes. For instance, to manage scalability, adaptation process based on the mechanism to increase resources allocated to the virtual machine (or to the ESB instance) can be applied first to perform the vertical elastic concept. And when not more resources can be added, the adaptation process to apply the horizontal elasticity is executed to deploy to a new virtual machine (or a new ESB instance) and to create a cluster (Figure 5.16).

Next section presents services developed to allow the execution of defined adaptation processes.

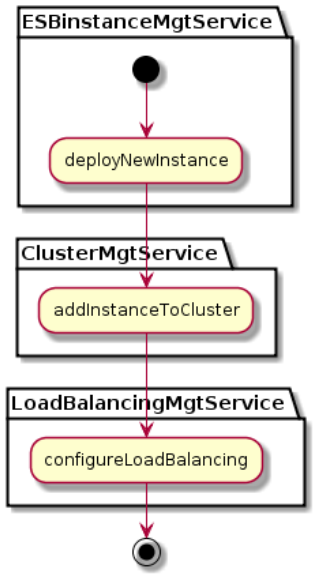


**Figure 5. 15 Complex plan process**

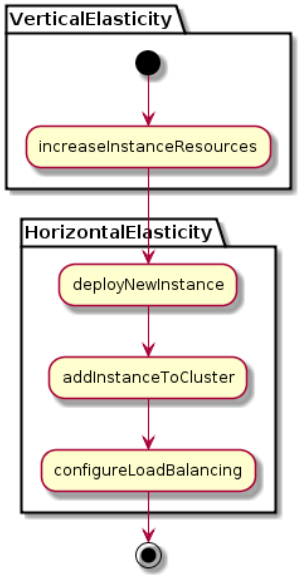


**Figure 5. 16 Composed complex plan processes**

### 5.3.5 Execution services

In the ASB context, several services are developed to control and coordinate the execution of mechanisms proposed to correct QoS and scalability degradation. These services are distributed in two families:

- the first family of services implements the orchestration defined by the adaptation process and controls the second family of services;
- the second family of services performs changes on the ESB and the computing resources, to modify partially or totally their states when symptoms such as QoS or scalability

degradation are detected in order to keep the ASB at the good performance level. They allow integrating the intra-bus and extra-bus mechanisms to take into account requirements and constraints of distributed systems. Their main actions are based on a structural and behavioural management of mechanisms [WAM 09]:

- structural adaptation: replace an implementation of a mechanism by another one, or more simply plugging in or out a mechanism that is respectively required or no longer needed;
- behavioural adaptation: modify some parameters of the already running mechanisms to change the way they behave during the integration.

Next figure 5.17 illustrates the interactions between the two families of services to apply the horizontal elasticity concept based on an increase of the cluster's instances.

### 5.3.6 Summary

The plan functionalities of the ASB are performed according to a set of predefined policies. Those policies can be defined following an expert system. In this thesis, a smarter way to define the plan policies is proposed with a model based on the semantic description of the potential actions that can be triggered when issues occur. The different actions are semantically expressed to provide a unified and detailed description. This allows defining high-level rules that guide the autonomic behaviour of the ASB. This approach is efficient and extensible since when a new corrective action is proposed, the mechanism has just to be characterized and instantiated in the semantic model in order to be taken into account.

Execution services are proposed for coordination of actions to be executed on the ESB or the computing resources during the mediation processes to deal with the QoS and scalability requirements based on the defined adaptation processes.

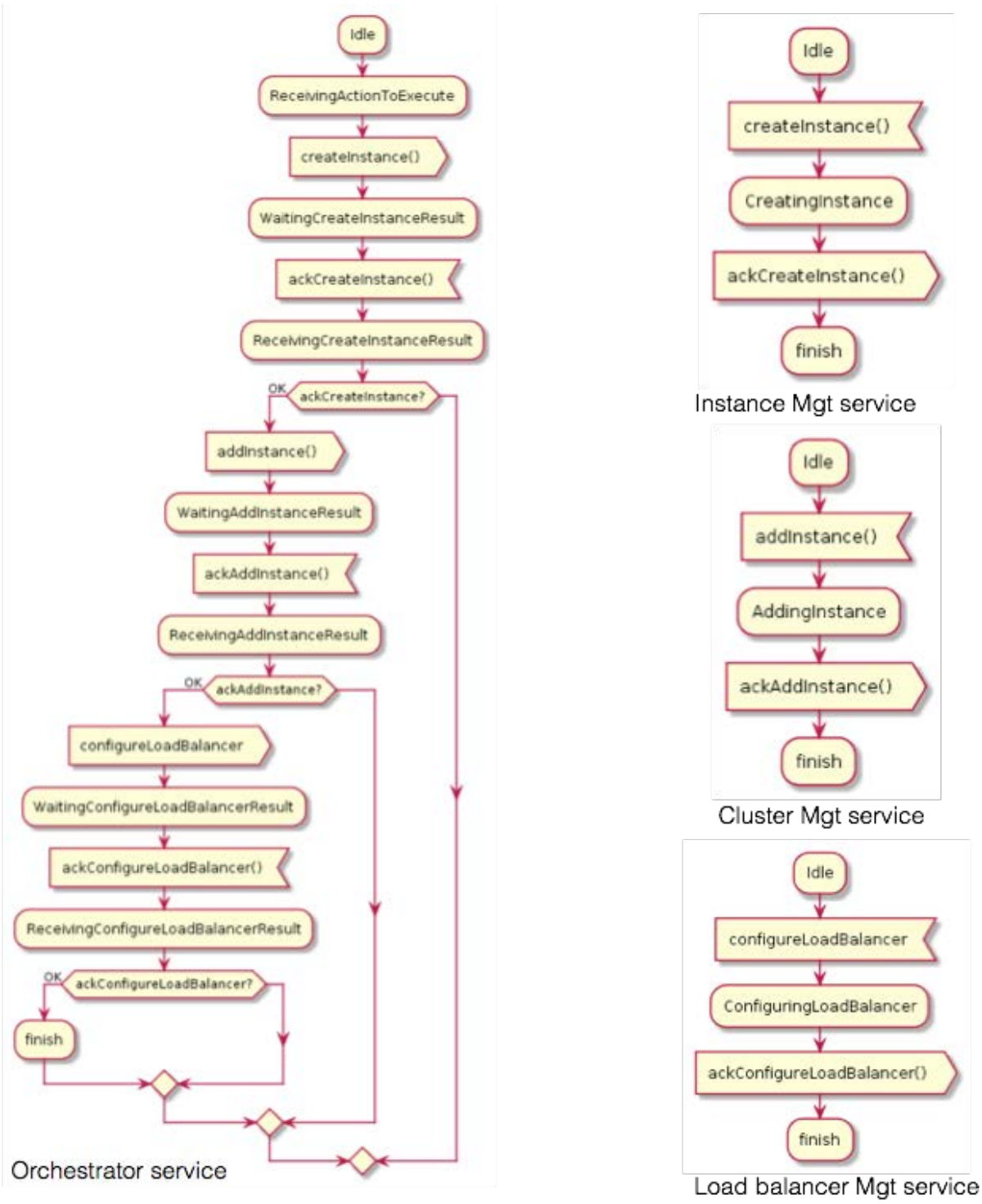


**Figure 5. 17 Interaction between distributed execution services**

## 5.4 Conclusion

In this chapter, we have presented the proposed approaches, solutions and techniques for the analysis of monitored data and symptoms coming from the Monitor component and the definition of the plan to be executed with the selection of the most adequate adaptation actions to apply in order to deal with potential issues.

We develop a probabilistic diagnostic model that offers solutions to analyze symptoms received from the Monitor component.

The model is generic and useful since it can be exploited for different businesses and distributed systems' characteristics. It just needs to be well trained and the EPES platform can be used to run several scenarios and exploit data traces. Improvement of the proposed probabilistic diagnostic model consists in solutions to define the symptoms (to be observed) that allow inferring the state of the whole ASB.

The output of the Analyze component is a specific request for change to guide the definition of the plan to be executed. The actions to be executed when symptoms are detected consist in the intra-bus and extra-bus mechanisms proposed to provide an adequate scalability and QoS to the supported integrations. These mechanisms can be more efficient and better suited in specific situations than others. A model to classify and characterize semantically all these mechanisms is proposed. The model allows the Plan component of the ASB to define, for each request for change received from the Analyze component, the best mechanism or composition of mechanisms that must be used.

In this work, the proposed adaptation processes are not exhaustive. Approaches to identify, characterize and define more complex adaptation processes associated to each mechanism can improve the model. Defined adaptation processes need to be associated to a solution that controls them to avoid faults during the execution. Solutions similar to the ones proposed in [IFT 12] and [IGL 13] can be integrated to control the execution of the workflows implementing the adaptation processes to prevent faults, deadlocks or continue oscillations and to ensure that defined plans allow achieving the goals.

Next chapter 6 introduces the IMAGINE European project [IMA 11] and presents an illustration of ASB usage in the IMAGINE context.

# Chapter 6. **Dissemination, Application and Evaluation**

**Contents**



## 6.1 Introduction

The Autonomic Service Bus (ASB) is proposed as a new communication middleware solution able to deal with integrability and interoperability by taking into account QoS and scalability requirements in an efficient and autonomic way.

Based on a JBI compliant ESB implementation deployed on a “cloud-oriented” infrastructure, the proposed ASB framework integrates a set of mechanisms that can be applied on the ESB or the underlying computing resources to manage QoS and scalability. Solutions implementing the autonomic computing framework are also included to ensure the control and coordination of these mechanisms in an autonomic way.

The aim of this chapter is to show how we evaluate and validate the ASB proposal in the context of the European project IMAGINE that illustrates well the second and third generations of distributed systems.

This chapter is structured as follows: section 6.2 introduces the IMAGINE project. The illustration of the ASB usage in the IMAGINE context is presented in section 6.3. Section 6.4 concludes the chapter.

## 6.2 The IMAGINE Project

The IMAGINE research project is a European project that targets the development and delivery of a new comprehensive methodology and a platform for effective end-to-end management of dynamic manufacturing networks (DMN) [IMA 11]. It is aimed at supporting the emergence of a powerful and new production model, based on community, collaboration,

self-organization and openness. This will allow boosting the productivity and competitiveness of small and mid-sized manufacturers (SMEs), which will become more responsive and agile in designing and producing new generation Future-Internet manufacturing applications and systems.

The main contributions of the project are the IMAGINE DMN lifecycle management methodology and the DMN ICT Platform that supports the proposed methodology (Figure 6.1).

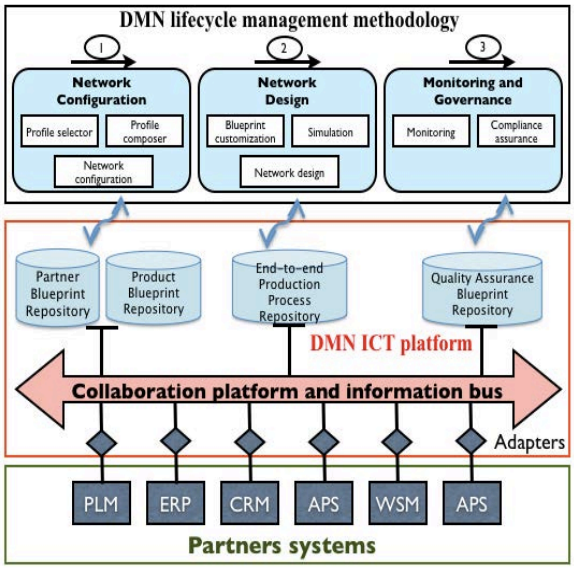


**Figure 6. 1 IMAGINE methodology and ICT platform**

The targeted methodology allows efficiently managing a networked supply chain in three phases: Network Configuration, Network Design, Monitoring and Governance. The Network Configuration phase aims at identifying and selecting partners for collaboration. During Network Design phase, the distributed systems of selected partners are interconnected and the end-to-end supply chains processes are put in place. The Monitoring and Governance phase aim at managing and controlling at runtime the operations of the network.

The ICT platform gives the technical solutions that effectively enable and concretely support the management of networked manufacturing supply chains. It is based on a set of blueprint repositories that introduce the followed data format model to describe the final wanted product, the partner's capacity, the end-to-end process and the needed quality. An ESB Service Bus integrates the distributed systems of collaborating partners involved in the different processes and allows the access to the blueprint.

The IMAGINE Project illustrates well the second and third generations of distributed systems since it allows the collaboration of several SMEs. The project demonstrations are based on five living labs (LL), each one with a specific market and business context:

- Aerospace and Defense domain (EADS LL)
- Multi-site single factory domain (IPA LL)
- Furniture Manufacturing domain (AIDIMA LL)
- Car Manufacturing domain (FIAT LL)
- Engineering domain (UoW/WMCCM LL)

Inside a living lab representing a set of inter-domain collaborations, enterprises have to expose internal processes, tools, services and data integrated through IMAGINE platform that is supposed to support in concurrence all the five living labs (Figure 6.2). Next section presents how the proposed ASB framework can be used in the IMAGINE context.

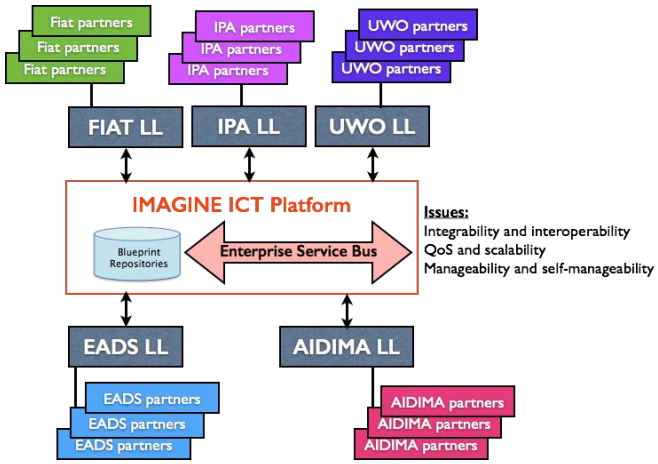


**Figure 6. 2 IMAGINE LL support**

## 6.3 ASB usage in Imagine context

### 6.3.1 Aerospace and Defense living lab

A deployment and evaluation of the ASB implementation have been done in the framework of the IMAGINE European project particularly through the integration of our solution in the Aerospace and Defence living lab environment. Within this living lab, AIRBUS, Engine Alliance, AIRCELLE, SNECMA, Rolls Royce are part of involved partners in the process of the aircraft production. The goals of this collaboration are to specify, design, produce and

support aircraft models composed by a nacelle, an engine, a reactor, a cabin and the wings (Figure 6.3).

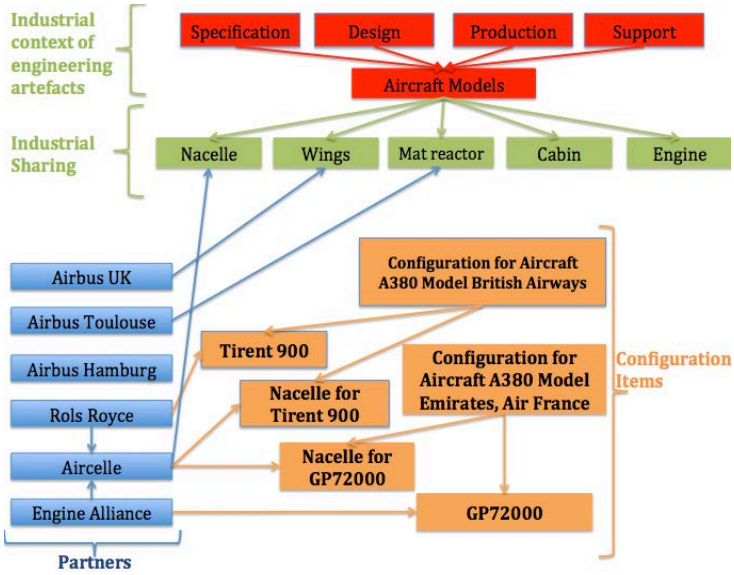


**Figure 6. 3 Distributed collaboration**

### 6.3.2 Collaborative network design and configuration through the ASB

The aerospace and defence living lab inter-enterprise collaboration results the integration of heterogeneous processes, tools, services and data. They publish and share heterogeneous data with a diversity of representation and format. And because they are in different fields, they can have a different perception and understanding of the exchanged data. Using the ASB, all these distributed systems can be connected to deal with the integrability and interoperability requirements (Figure 6.4). For example, merely for the design part, several partners are involved in designing one or several parts of the models, and they can expose their Computer-aided Design (CAD) or Product Data Management systems as services through the ASB (Figure 6.5).

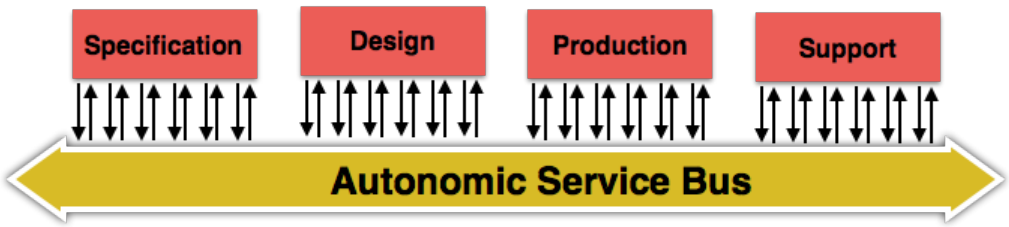


**Figure 6. 4 ASB-based aircraft model lifecycle support**

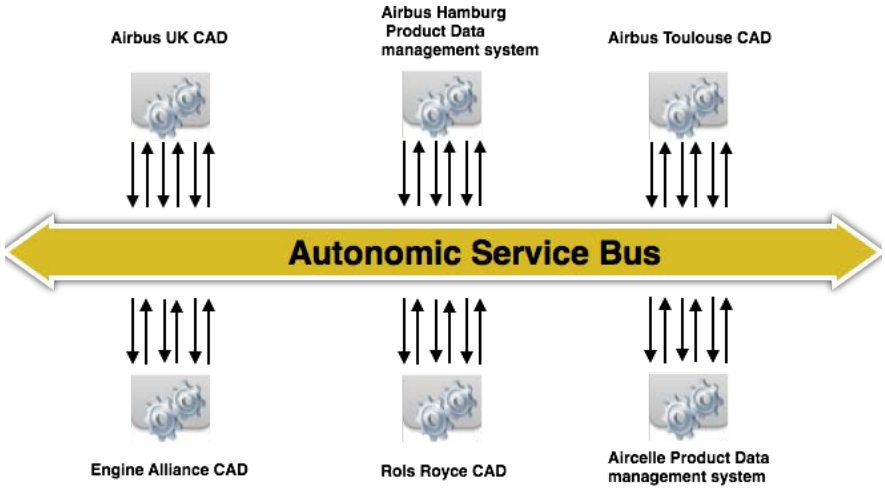


**Figure 6. 5 Design systems integration**

Therefore, the ASB is able to satisfy the integrability and interoperability properties of involved distributed systems.

A process can be designed to describe the interactions between distributed systems of all the partners and the needed mediation between these systems. The process can be implemented using the Business Process Execution Language (BPEL) and can be deployed on the ASB. This BPEL process will be invoked at each time a collaboration request is sent.

During the network operations, the ASB is able to support a large number of parallel and concurrent BPEL processes representing collaborative productions with different priorities (classes of services to guide the production). In this context, a high volume of communications that have different QoS requirements are managed by the ASB that implements adequate strategies to provide dynamically the required QoS and scalability.

Next section gives a network management example to satisfy QoS and scalability issues.

### 6.3.3 Collaborative network management through the ASB

The proposed ASB is an important contribution in the IMAGINE project since it tackles QoS and scalability degradations by offering mechanisms and services for a smart and autonomic management of QoS and scalability requirements.

Indeed, during the network operations, manufacturing processes are monitored; anomalies are predicted and avoided, or detected and repaired. For instance, when a service of a partner receives too many requests, congestion is predicted and the ASB enforces flow control,

shaping or load balancing strategies. The ASB also monitors the state of the underlying computing resources. Being aware of the real time overload, the ASB enforces adequate corrective strategies based on the extra-bus mechanisms.

Using the proposed Emulation Platform for ESB System (EPES) to illustrate this management of QoS and scalability issues during network collaboration, a set of services interactions and mediation processes similar to what we can have through the IMAGINE platform are simulated. A BPEL process orchestrating these services is implemented and deployed (Figure 6.6).

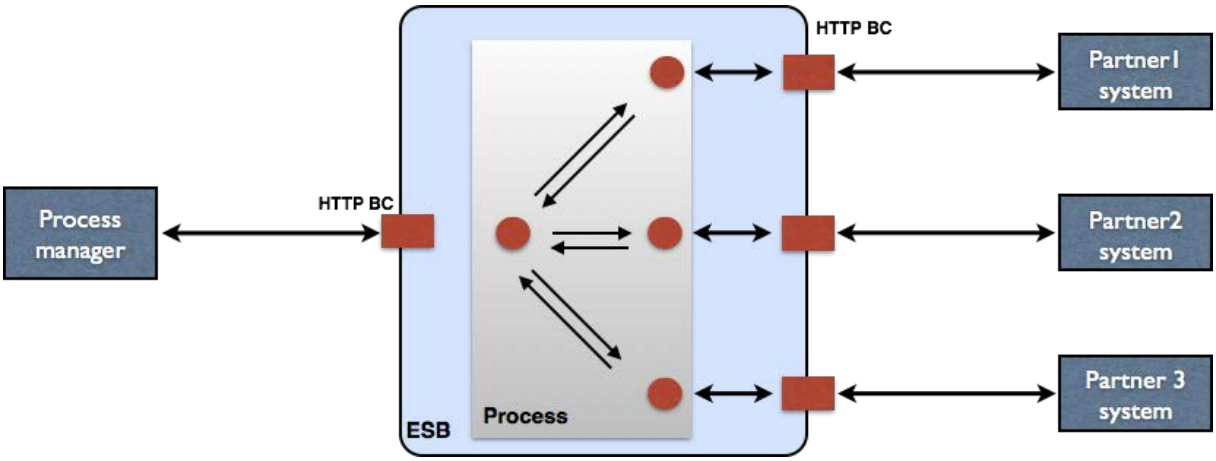


**Figure 6. 6 Process scenario illustration**

Like similar distributed systems design and deployment, we assume that the platform is initially dimensioned and deployed based on a predefined business plan and initial exchange traffic between interconnected systems. The EPES components (the different virtual machines that host the ESB and the web services) were firstly deployed in a physical server with the following configuration (Table 6.1):

| Configurations | Server characteristics | | | |
|---|---|---|---|---|
| | Processor | Memory | Disk | OS |
| **Initial Configuration** | 1,8 GHz Intel Core i7 | 4 Go 1333 MHz DDR3 | 250 Go SSD | OS X 10.9 (13A603) |

**Table 6. 1 Initial server characteristics**

Something that often happens is that the platform will not be able to support the increased workload extending the initial plan and could start presenting QoS and scalability issues.

For instance during a high production process, several parallel and concurrent transactions need to be supported by the IMAGINE platform. This situation can lead to several issues such

as service unavailability, high response time, decrease of reliability, etc. The figure 6.7 shows the evolution of the manufacturing process response time according to the number of concurrent transactions (processes) in the system. The figure shows that the response time can become unacceptable (14s) since it increases with the number of transactions.

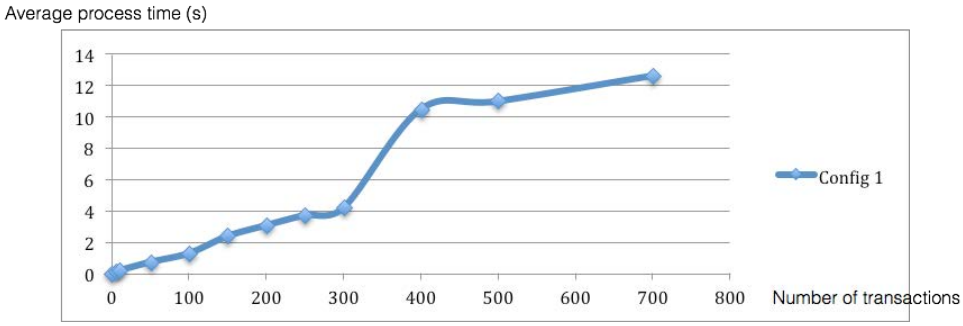


**Figure 6. 7 Process response time evolution**

We used these results to illustrate an easier autonomic reconfiguration scenario driven by the load of the physical server used for the deployment. The strategy is to dynamically migrate the virtual machines of the platform from the initial server to another when the resource usage of the initial server is critical. The table 6.2 presents the configurations of the initial and second server.

| Configurations | Server characteristics | | | |
|---|---|---|---|---|
| | Processor | Memory | Disk | OS |
| **Initial Configuration** | 1,8 GHz Intel Core i7 | 4 Go 1333 MHz DDR3 | 250 Go SSD | OS X 10.9 (13A603) |
| **Second Configuration** | 2,6 GHz Intel Core i7 | 16 Go 1600 MHz DDR3 | 2 To HDD | OS X 10.9.3 (13D65) |

**Table 6. 2 Second server characteristics**

To implement the strategy, we define a threshold that does not have to be reached by the CPU usage of the hosting machine (*maxUsageCPUThreshold* at 90%) and use the monitoring services to observe both the ESB performance and the CPU.

The monitoring services are lightweight and do not impact the running scenario. For instance the JBI monitoring service used to get statistics related to communications through the ESB has a negligible impact on the IT resources. The study shows that the impact is insignificant (Figure 6.8). The CPU stays “intact” (less than 1%), as the monitoring does not require processing operations; however we notice a small oscillation of the memory.

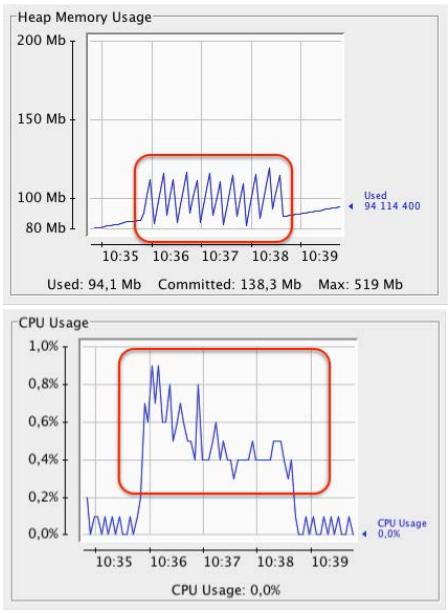


**Figure 6. 8 Evaluation of the JBI monitoring service**

Similarly the JVM monitoring service used to get the CPU usage of the running platform has an "acceptable" impact on the integration performance due to the usage of the CPU and memory resources by the agent deployed at the ESB side (Figure 6.9). The figure 6.10 shows that the impact of the JVM monitoring on the communications (propagation time) is negligible.

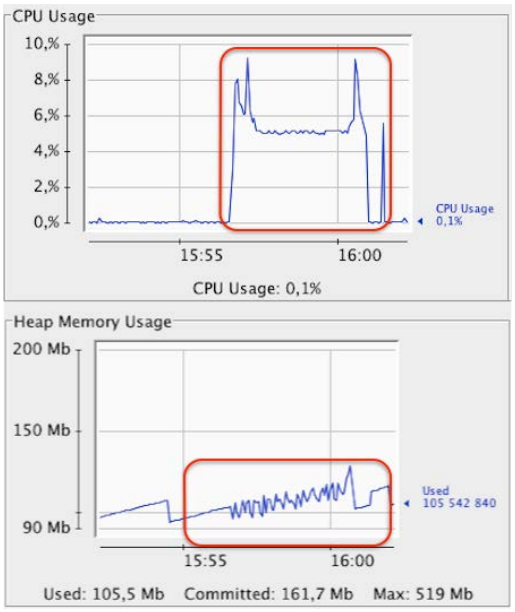


**Figure 6. 9 Evaluation of the JVM monitoring service impact on the resources**

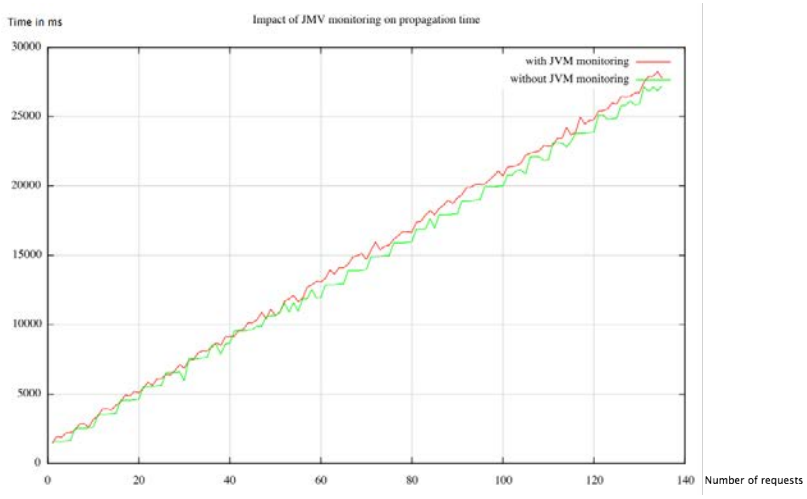


**Figure 6. 10 Evaluation of the JVM monitoring service impact on the communication**

Using the monitoring services, the CPU load is compared to the defined threshold (maxUsageCPUThreshold). When the maxUsageCPUThreshold value is reached, the Monitor component sends a symptom detection to the Analyze component. The diagnostic of the Analyze component is correlated the detection to a high number of concurrent transactions and a lack of resources. The Analyze component sends a degradation alarm to the Plan component. Once the Plan component receives the alarm, it triggers the migration mechanism that is enforced by the Execute component.

The next figure 6.11 and table 6.3 show how the process response time is improved by migrating the virtual machines to a physical host with better capacity when the number of transactions reaches 300. The time spent by the mechanism to perform its tasks depends on the number of virtual machines to migrate and their loads. However, this time does not have an impact on the running business as the migration is transparent and done in live.

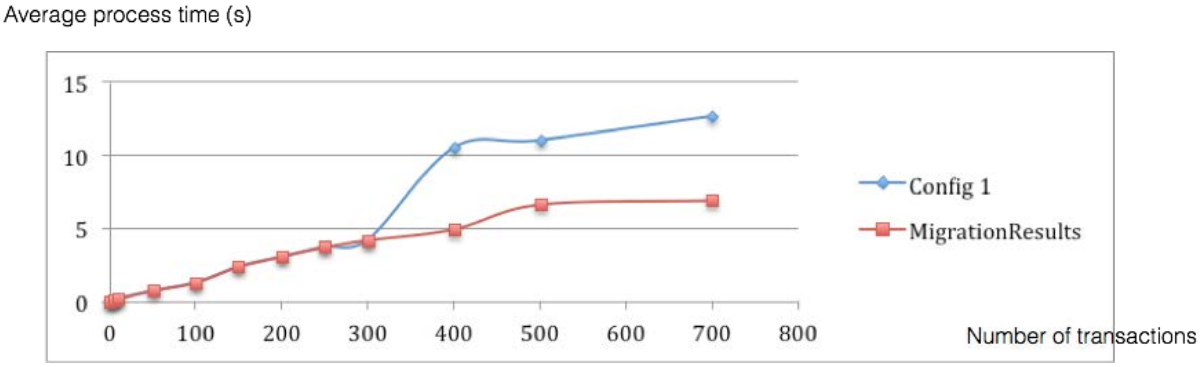


**Figure 6. 11 Process response time evolution comparison**

| Number of transactions | Process time (average in seconds) | |
|---|---|---|
| | Config 1 | Migration |
| **0** | 0 | - |
| **5** | 0,09 | - |
| **10** | 0,18 | - |
| **50** | 0,72 | - |
| **100** | 1,283 | - |
| **150** | 2,273 | - |
| **200** | 3,069 | - |
| **250** | 3,714 | - |
| **300** | 4,184 | - |
| **400** | 10,482 | 4,92 |
| **500** | 10,999 | 6,632 |
| **700** | 12,624 | 6,915 |

**Table 6. 3 Process time results comparison with two configurations**

## Conclusion

This chapter has presented how contributions of this thesis have been used in the context of the European project IMAGINE that illustrates well the second and third generations of distributed systems.

This demonstration validates our position about the needs to introduce the Autonomic Service Bus as the solution that helps to integrate and support the communication of distributed, heterogeneous and pervasive systems by taking into account QoS and scalability requirements.

The EPES platform is used to run scenarios introduced in the IMAGINE project and to show improvement thanks to a dynamic reconfiguration.

# Chapter 7. **Conclusions and Perspectives**

**Contents**



The development of Internet technologies and the current economical and industrial worldwide context have more and more allowed the evolution of largely IT distributed systems. In this thesis, three generations of distributed systems have been identified together with a set of requirements that need to be taken into account by the underlying communication layers. The first generation presents QoS and scalability requirements; the manageability and self-manageability of the communication solution to satisfy these requirements depend on only two interacting entities. The satisfaction of these QoS, scalability, manageability and self-manageability requirements is more complex for the second generation, since integrability and interoperability also need to be taken into account and the interacting entities are more than two. Finally, the characteristics of the third generation (e.g. multi-domains, dynamicity of the context, etc.) make even more complex the satisfaction of QoS, scalability, integrability, interoperability, manageability and self-manageability requirements.

The management of these requirements is now made possible thanks to the rich functionalities provided by the different protocols and services available at the different layers of the communication systems. Several protocols and services were proposed at the network and transport OSI layers to satisfy users requirements and to take advantage of new network technologies (e.g. IntServ [BRA 94], DiffServ [NIC 98], MPLS [RFC 3031], DCCP [KOH 06], SCTP [STE 07], MPTCP [FOR 10], etc.). These protocols and services were proposed to better cope with QoS requirements, such as order and reliability for the traditional file transfer or email applications, or guaranteed bandwidth and bounded delay for video/audio streaming or conferencing multimedia applications. The integrability, interoperability, QoS, scalability, manageability and self-manageability were also (partially) developed within middleware-level solutions designed to facilitate the interconnection of largely distributed systems (compared to conventional solutions based on TCP sockets), and especially to allow taking into account applications and data sources distribution and heterogeneity (e.g. Remote Procedure Calls

[BIR 84], Distributed Objects [COM 13] [MIS 13] [RMI 13] [COR 13], Message Oriented Middleware [CUR 04], Event Driven Architecture [GAR 13], Resource Oriented Architecture [FIE 00] or Service Oriented Architecture [SOA 06]).

**This thesis enhances the existing communication middleware solutions** to better satisfy the requirements of distributed systems. We propose an Autonomic Service Bus (ASB) as an extended Enterprise Service Bus with the ability to deal with the integration and the interoperability of services based distributed systems. The ASB also satisfies the scalability and QoS demands in an autonomic way.

To guarantee the QoS and scalability requirements, a lot of mechanisms extend the ESB implementations and the underlying computing resources. Self-configuring and self-adaptation capacities are provided to control the complexity of its manageability.

All these properties give to the proposed ASB the position of a new generation of integration and communication middleware for the future Internet. An illustration has been done in the context of the IMAGINE project, which is an European that illustrates the second and third generations of distributed systems by proposing new approaches, methodologies and technical solutions to build flexible, agile and dynamic collaboration of networked enterprises. We demonstrate in this thesis how the ASB can be used to manage integrability, interoperability, QoS, scalability, manageability and self-manageability required in the IMAGINE context.

The following sections present a summary of the contributions of this thesis, the conclusion of the work and several research perspectives.

## 7.1 Summary of contributions

After a general introduction of the different generation of distributed systems and their requirements in terms of integrability, interoperability, QoS, scalability, manageability and self-manageability, we have presented in **chapter 2** a large study of protocol, paradigms, platforms, frameworks, technologies and research works that deal with these requirements.

The evolution of the various generations of middleware solutions has been presented. Works and proposals to extend ESB solutions for QoS management have been detailed with a survey of advanced ESB having enhanced features to deal with requirements defined for new generation of communication infrastructures. The lacks to be covered have been identified to

position our thesis proposal since the study of ESB improvements proposals allowed us justifying the introduction of the ASB as an ESB with the ability to manage in an autonomic way the QoS and scalability.

The architectural framework of the proposed ASB is presented in **chapter 3**.

Several mechanisms for the management of QoS and scalability requirements have been proposed. They are classified in two categories: “intra bus” and “extra bus” mechanisms:

- the “intra bus” mechanisms target the ESB implementation. They are inspired from our background in transport and network-oriented solutions for QoS management (e.g. congestion control, error control, shaping, differentiation, resource reservation, admission control, etc.);
- the “extra bus” mechanisms target the underlying resources level (e.g. JVM, virtual and physical infrastructure). They are based on virtualization and cloud-computing characteristics (e.g. clustering, federation, load balancing, elasticity, self-provisioning, live migration, etc.) to avoid the overprovisioning or oversized techniques and to manage well resources of the IT infrastructures.

The autonomic computing framework has been implemented to ensure the control and coordination of these mechanisms in an autonomic way.

Several architectural design requirements come from the ASB proposal. We have followed a methodology based on the model-driven architecture (MDA) approach for a generic architecture to be implemented using any existing ESB implementations. We have adopted the Java Business Integrator (JBI) standard and specification proposed by the Java Community Process (JCP) in the JSR 208 specification [TEN 05] to have an open and extensible solution. JBI defines a platform for building ESBs using a set of plug and play services components. It allows integrating the proposed QoS and scalability mechanisms as a set of pluggable components.

An Emulation Platform for ESB Systems (EPES) has finally been designed, implemented and used towards the implementation of the ASB. EPES allows simulating the behaviour of distributed systems (services consumers and providers) that communicate through a real ESB implementation. EPES also allows assessing ESB systems by identifying and characterizing

their limits. The implemented version of EPES is the starting point of the ASB implementation

The **chapter 4** has introduced the step that consists in dealing with the manageability of all the basic components (software and hardware) that constitute the ASB. The chapter has detailed the proposed monitoring solutions able to supervision all the operational levels that constitute the ASB. Several monitoring services have been developed to not only supervise the ESB level, but also the underlying infrastructure resources level (e.g. Java Virtual Machine, virtual machines and physical server on which run the ESB, etc.). Different approaches to develop these monitoring services for the different levels have been studied to measure their cost and to choose the ones in order to limit the impact on the ESB and the distributed systems operations. An event-processing module has been included in addition to the monitoring services for the treatment of the monitored data in order to recognize and detect complex patterns correlated to potential symptoms. Methodologies to identify, define and characterize symptoms that need to be detected by the event-processing module have also been proposed.

The **chapter 5** has presented the solutions developed to build a knowledge base needed for the analysis of monitored data and symptoms, and for the definition of the adequate plan to execute.

A first section has presented the application of probabilistic techniques to develop an extensible model for a reliable and smart analysis of symptoms. The models are designed and used to diagnose symptoms. They help to make the diagnostic and to have a global view of all the parameters of the ASB based on the value of part of them represented by the symptoms.

A second section has presented the development of model and rule-based module that guide the choice of the adaptive plans to deploy when anomalies are detected. The models are used to correlate a diagnostic to a corrective solution by helping to define the best mechanism or composition of mechanisms that must be used in front of detected symptoms.

The final part of chapter 5 has detailed the execution services proposed for coordination of actions to be executed on the ESB or the computing resources during the mediation processes to deal with the QoS and scalability requirements based on the defined adaptation processes.

The **chapter 6** has presented how we have evaluated and validated the ASB proposal in the context of the European project IMAGINE that illustrates the second and third generations of distributed systems. We have shown how the ASB can be used to apply collaborative network design and configuration and how the ASB can tackle QoS and scalability degradations that can occur in the IMAGINE platform.

The results obtained using the ASB allow concluding that the integrability, interoperability, QoS, scalability, manageability and self-manageability requirements may be covered by the new generation of communication middleware that we propose within the ASB. Indeed:

- the proposed intra-bus and extra-bus are well designed to respond to potential QoS and scalability degradations and results of the ones we implement show well justifications of their introduction and improvements coming from them;
- the different solutions proposed to implement the autonomic manager components for the self-manageability satisfaction show that:
    - the monitoring services do not have an impact on the running businesses; that the probabilistic model allows to diagnose the state of the whole system based on received symptoms from the Monitor component;
    - the semantic model for the plan helps to define the adaptation process based on the mechanism or composition of mechanisms to be used in front of detected symptoms;
    - finally, the application of adaptation processes using executions services improve the QoS and scalability.

## 7.2 Perspectives

The main **short-term perspectives** identified from this thesis are:

*Extension of proposed architecture*

The proposed architecture for the ASB results in the extension of an ESB and its underlying computing resources with QoS and scalability mechanisms and an autonomic manager designed to manage the coordination and use of the mechanisms in a smart and efficient way. It can be improved by integrating other existing middleware platforms (as the MOM solution) that will help to better implement the proposed mechanisms (for instance by allowing to cache messages).

*Self-adaptive monitoring solution*

Several monitoring services for the data and metrics collection have been proposed. A way to control these services in order to make them self-adaptive will improve the scalability and the efficiency of the monitoring solution. The goal will be to reduce the cost of the monitoring and the quantity of data to be processed by the event-processing component by dynamically activating or deactivating part of the monitoring services in order to collect only data related to symptoms to detect or to predict. The proposed probabilistic diagnostic model will give the capability to infer the state of other parameters from the symptoms.

This model can be improved by covering a big challenge that consists in defining the most significant parameters to be observed according to the symptoms that allow inferring the state of the whole ASB. Indeed, it is needed to have a solution that helps to identify the critical parameters and variables that need to be observed according to their monitoring cost but also according to their usability to infer the other parameters.

*Adaptation plan processes extension*

The models are used to correlate a diagnostic to a corrective solution by helping to define the best mechanism or composition of mechanisms that must be used in front of the detected symptoms. In this thesis, the proposed adaptation processes are not exhaustive. Approaches to identify, characterize and define more complex adaptation processes associated to each mechanism would improve the model. Defined adaptation processes need to be associated to a solution that controls them to avoid faults during the execution. Solutions similar to the ones proposed in [IFT 12] and [IGL 13] can be integrated to control the execution of the workflows implementing the adaptation processes to prevent faults, deadlocks or continue oscillations and to ensure that defined plans allow achieving the goals.

Among the **long-term perspectives**, two of them are selected to guide our future research works namely:

*The Autonomic Service Bus as a Service (ASBaaS)*

With the development of distributed systems, Gartner preconized in 2011 the needs of Cloud services integration as potential alternatives to traditional communication solutions *(Integration Platform as a Service - iPaaS)* [PER 11] to simplify both development and use of infrastructures and platforms intended to deploy and scale the resources required for

interconnecting distributed systems within the same organization as well as across multiple organizations. Among the advantage of these solutions, we can also notice the externalization and the self-provisioning of resources that allow avoiding the complex management and the overprovisioning when the infrastructure is managed internally within the organizations.

In the thesis, the virtualization and Cloud computing characteristics have been exploited to satisfy the QoS and scalability requirements of distributed systems (e.g. the dynamic provisioning and the elasticity of the computing resources, the dynamic management of the number of ESB instances and the topology deployment model, etc.). The adoption of these concepts and the deployment of the ASB solution in a "cloud-oriented" environment give the basis to go toward an *Autonomic Service Bus as a Service* (*ASBaaS*) proposal as an efficient and advanced communication and integration solution available at the *PaaS layer (Platform as a Service)* of the Cloud Computing.

*The Autonomic Manager as a Service (AMaaS)*

Nowadays, autonomic systems are more and more developed to reduce the complexity of systems management. The way we design and implement the Autonomic Manager used for the Autonomic Service Bus can be followed and extended to go towards an *Autonomic Manager as a Service* (*AMaaS*) available as a service in the cloud.

Indeed, in this thesis, to have a scalable autonomic manager without integrability and interoperability issues, we follow principles promoted by the SOA paradigm (Figure 7.1).

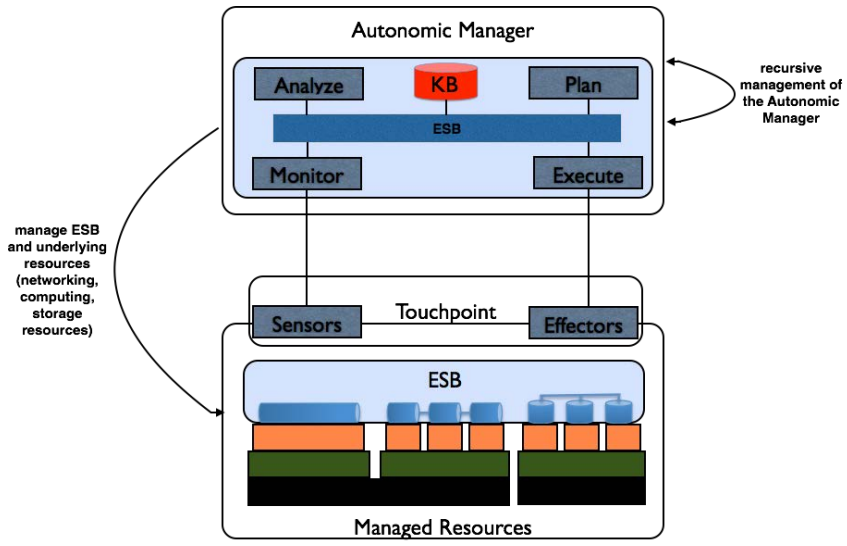


**Figure 7. 1 Service-Oriented Autonomic Manager**

The different features of the autonomic manager are designed as a set of services interconnected by an ESB. By this way, the integrability and interoperability of techniques, tools and solutions that can be used for Monitor, Analyze, Plan and Execute functions can easily be managed. Moreover, deploying and interconnecting the Monitor, Analyze, Plan and Execute components as a set of services using an ESB allow managing the scalability of autonomic manager by following the same approach proposed to develop the ASB.

This approach can be extended to go towards an *AMaaS* based on the instantiation of an ESB integrating all the Monitor, Analyze, Plan and Execute functions. The *AMaaS* will support several monitor services, decision models, and adaptation strategies. It will be configured and instantiated according to the system to manage but also the policies that guide the autonomic behaviour.

# Résumé en français

***Contexte***

L'évolution des technologies de l'Internet, avec des supports de communication filaires ou sans fil de plus en plus à haut débit (ADSL, fibre optique, 3G, 4G, etc.), et des terminaux utilisateurs fixes ou mobiles conçus pour tirer mieux profit de ces capacités, a fortement révolutionné le monde des applications distribuées.

En fonction de la complexité de ces applications, et en particulier de la distribution et de l'hétérogénéité des entités qui les composent, trois générations d'applications distribuées ont été identifiées :

- une première génération basée sur le modèle « Client-Serveur » et caractérisée par l'interaction de deux entités ;
- une deuxième génération basée sur le modèle « multi-tiers » et caractérisée par l'interaction de plus de deux entités au sein d'une organisation ou entreprise ;
- et une troisième génération également basée sur le modèle « multi-tiers » et caractérisée par l'interaction de plus de deux entités distribués, mais au sein de plusieurs organisations ou entreprises.

L'étude et l'analyse de ces trois générations nous ont permis d'identifier un certain nombre d'exigences que les solutions de communication doivent satisfaire. Parmi ces exigences, celles que nous avons considérées dans cette thèse sont définies comme suit [ITU 97] [EEL 05] [ITU 08] [IHR 13] [NOR 13]:

- ***la Qualité de Service*** définie comme la capacité de la solution à effectuer les transmissions en respectant un certain niveau de :
  - ***fiabilité*** définie comme la capacité à assurer la transmission sans perte, ni duplication, ni désordre;
  - ***débit*** défini comme le nombre de messages et de transactions que la solution de communication peut traiter en une période donnée ;
  - ***latence*** définie comme le temps aller-retour entre la date à laquelle les messages sont envoyés et la date de réception des réponses;
  - ***disponibilité*** définie comme la capacité à rester dans un état opérationnel à tout instant.

- ***scalabilité*** définie comme la capacité à supporter l'évolution des utilisateurs en garantissant un même niveau de QdS ;
- ***intégrabilité*** des entités distribuées, définie comme la capacité de la solution à agréger facilement un nouveau composant à une application existante ;
- ***interopérabilité*** définie comme la capacité de la solution à rendre compréhensible et exploitable les messages échangés entres les entités distribuées de l'application ;
- ***manageabilité*** définie comme la possibilité de superviser et de contrôler la solution de communication ;
- ***auto-manageabilité*** définie comme la capacité de la solution de communication à pouvoir se superviser et se contrôler de façon autonome.

***Problème***

Pour suivre l'évolution technologique, répondre aux exigences que les solutions de communication doivent satisfaire et faciliter le développement des applications distribuées, des middleware de communication et plusieurs protocoles de niveaux réseau et transport (au sens OSI) ont été proposés.

Les protocoles et architectures de niveaux réseau et transport offrant des solutions de gestion de la QdS ont permis de mieux supporter les applications classiques (ex. transfert de fichier, échange de mail, etc.), mais aussi les applications plus contraignantes dites multimédias (ex. audio ou vidéo conférence, etc.). Parmi ces protocoles et architectures, nous avons étudié : IntServ [BRA 94], DiffServ [NIC 98], MPLS [RFC 3031], DCCP [KOH 06], SCTP [STE 07], MPTCP [FOR 10].

Toutefois :

Ces protocoles/architectures ne sont pas largement déployées actuellement, soit parce qu'ils présentent des limites en termes de scalabilité [YEN 04], soit parce qu'ils ne sont pas acceptés par certains équipements présents dans l'Internet [JIA 05]. Leur diversité limite l'intégrabilité en rendant complexe le travail du développeur qui a besoin d'une grande expertise et d'une large connaissance de l'ensemble des protocoles existants s'il utilise directement l'interface de la couche transport. Pour finir, aucun de ces protocoles standards n'est conçu pour satisfaire l'interopérabilité ou pour changer de comportement en fonction du contexte (manageabilité ou auto-manageabilité).

La croissance des middlewares de communication a, quant à elle, facilité le développement de nouveaux modèles d'applications distribuées en permettant de satisfaire les besoins d'intégrabilité et d'interopérabilité des entités qui composent ces applications. Parmi ces middleware de communication, nous avons étudié: l'Appel de Procédures Distants (Remote Procedure Calls) [RFC 1057] [RFC 5531], les Object Distribués (Distributed Objects) [COM 13] [MIS 13] [RMI 13] [COR 13], les Middleware Orienté Messages (Message Oriented Middleware), les Architectures Guidée par les Evénements (Event Driven Architecture) [GAR 13], les Architectures Orientée Ressources (Resource Oriented Architecture) [FIE 00] et les Architectures Orientée Services (Service Oriented Architecture) [SOA 06].

Toutefois :

Les middleware de communication présentent toujours des limites pour garantir de manière efficace la QdS et la scalabilité [DAV 04] [ABI 09] [GAR 09] [RYA 11]. La scalabilité est un grand challenge vu l'évolution du nombre de systèmes à intégrer (100 billions de terminaux en 2015) et le volume de données échangées (42070 hexa-byte) [ISS 11]. Les solutions actuelles pour répondre à ces besoins restent des solutions statiques et pas optimales basées sur des techniques de redondance et de sur-provisionnement [OTH 01] [WSO 11].

***Positionnement***

Dans ce contexte, les objectifs majeurs de cette thèse sont d'améliorer les middleware de communication (capable d'assurer l'intégrabilité et l'interopérabilité) en leur dotant de capacité de gestion de la QdS et de la scalabilité.

L'étude des middlewares de communication existants nous a permis d'identifier le paradigme **d'Architecture Orientée Services (SOA)** comme l'approche la plus adoptée pour la satisfaction de l'intégrabilité et l'interopérabilité des systèmes distribués [THE 03] [CHE 06] [SOA 06] [EXP 14]. En suivant le paradigme SOA, les applications distribuées sont conçues comme une composition de services distribués qu'il faut intégrer pour mettre en place des processus plus complexes.

Parmi les solutions pour mettre en place une architecture orientée services, les **bus de services (Enterprise Service Bus – ESB)** ont été identifiés comme la solution technologique la plus mature qui permet l'intégration et la médiation de services distribués [CHA 04].

Exploitant des standards, les ESB implémentent des stratégies d'intégration et de médiation qui permettent la découverte et l'invocation des services distribués, la composition des services, le routage des messages, la transformation des messages, la conversion des protocoles de communication, etc. Les ESB assurent aussi l'intégrabilité et l'interopérabilité de middleware de communication basés sur des approches et des technologies hétérogènes.

Toutefois, bien qu'ils permettent l'intégrabilité et l'interopérabilité, les ESB peuvent présenter des problèmes de QdS et de scalabilité lorsqu'ils ont à supporter un grand nombre de transactions.

**Nous avons donc porté nos contributions sur les ESB** avec comme objectif d'enrichir les ESB existants avec des solutions leur permettant de garantir de manière efficace la QdS et la scalabilité.

***Contributions***

**Notre première contribution** a été la proposition de mécanismes pour la gestion de la QdS et de la scalabilité. Deux approches complémentaires ont été explorées :

- dans la première approche, nous nous sommes inspirés des mécanismes proposés au niveau des couches réseau et transport pour la gestion de la QdS et la scalabilité comme le contrôle de congestion, le contrôle d'erreur, la mise en forme du trafic, la différentiation, la réservation de ressources, le contrôle d'admission, etc. Ces mécanismes qui ont été développés pour venir étendre l'implémentation des bus de services sont classés comme des mécanismes « intra-bus ».
- dans la seconde approche, les mécanismes proposés sont basés sur les caractéristiques et les propriétés de la virtualisation et du Cloud Computing (ex : cluster, fédération, partage de charge, élasticité, auto-provisionnement, migration, etc.). Ces mécanismes, qui visent surtout une gestion efficace et intelligente des ressources infrastructurelles (ex : CPU, thread, mémoire, stockage, etc.) utilisées par le bus pour son fonctionnement, sont classés comme des mécanismes « extra-bus ».

**La deuxième contribution** vise à proposer une architecture de déploiement nécessaire pour la gestion, la coordination et l'exécution des mécanismes en fonction de chaque contexte. Nous avons proposé une architecture dynamique qui permet de couvrir les besoins en termes de manageabilité et de auto-manageabilité.

Pour doter le système de propriétés de supervisision et de contrôle de façon autonome, nous avons implémenté le cadre de l'**Autonomic Computing** introduit par IBM [IBM 05].

Avec le développement des mécanismes proposés et l'implémentation du cadre de l'Autonomic Computing, nous avons mis en place un ESB nommé **Autonomic Service Bus (ASB)**, qui a la capacité de gérer de façon autonome la QdS et la scalabilité.

L'ASB est constitué :

- des instances d'ESB, des ressources infrastructurelles utilisées par le bus pour son fonctionnement et des mécanismes proposés pour la gestion de la QdS et de la scalabilité. Ces différents composants sont considérés comme des éléments à superviser et à contrôler afin de pouvoir modifier ou adapter leurs comportements ou leurs configurations ;
- d'un contrôleur associé aux différents composants qui sont considérés comme des éléments à superviser et à contrôler. Le contrôleur est capable de superviser leurs différents états, d'analyser les résultats, d'identifier si une adaptation est nécessaire ou pas, de planifier les actions d'adaptation et de les mettre en œuvre.

**La troisième contribution** est une implémentation de l'architecture proposée.

Une plateforme d'émulation a été développée et utilisée tout au long de l'implémentation.

Le déploiement et l'évaluation de l'implémentation ont été effectués dans le cadre du projet européen IMAGINE proposé pour la mise en place des réseau dynamique d'usines et de manufactures du futur [IMA 11].

***Structure et résumé des chapitres***

L'organisation de la dissertation est faite comme suit :

**Le chapitre 1** introduit l'évolution des systèmes distribués. Les propositions de middleware de communication et de protocoles réseau et transport ont été analysées pour démontrer leurs limites. Cette analyse nous a permis de positionner notre proposition et d'introduire nos différentes propositions.

**Le chapitre 2** détaille l'état de l'art. Dans le but d'améliorer les middlewares de communication, une première partie du chapitre présente leurs évolutions. Nous montrons

ensuite comment chacune des approches facilite ou pas le développement des systèmes distribués.

De nos jours, la conception et le développement des applications distribuées suivent plus le paradigme d'Architecture Orientée Services. Ce paradigme permet de mieux garantir la distribution et l'hétérogénéité des entités qui composent les applications distribuées.

L'étude des différentes solutions techniques pour mettre en place une Architecture Orientée Services (serveurs d'application, middleware orienté message, intégrateur d'applications d'entreprise – EAI, bus de service – ESB) permet de conclure que les ESB représentent la solution technologique la plus mature qui permet l'intégration et la médiation de services distribués [CHA 04].

Toutefois, bien qu'ils permettent l'intégrabilité et l'interopérabilité, les ESB peuvent présenter des problèmes de QdS et de scalabilité s'ils doivent supporter un grand nombre de transactions. Ceci justifie la deuxième partie du chapitre, plus spécifique à notre proposition, qui présente les travaux qui ont cherché à améliorer les ESB.

Plusieurs propositions ont été faites pour la gestion de la QdS et de la scalabilité, généralement basées sur des techniques de redondance, de partage de charge, ou de sur-provisionnement des ressources. Des mécanismes ont été aussi proposés pour améliorer le routage offert par l'ESB ou limiter l'utilisation de services surchargés.

Des solutions ont été proposées pour introduire une gestion dynamique et manageable avec l'introduction de solutions de supervision et d'adaptation. Le plus souvent, elles visent la supervision de l'ESB, des services déployés, des messages échangés dans le but d'adapter dynamiquement le routage.

La limite des propositions gestion de la QdS et de la scalabilité est que les solutions sont souvent statiques, dédiées soit à la gestion de la QdS, soit à la gestion de la scalabilité et pas les deux à la fois. Et dans la gestion de la QdS, les ESB étendus proposés prennent en compte un seul paramètre (ex : soit la latence, soit la fiabilité).

Il existe aussi un manque d'une architecture qui intègre des mécanismes de supervision et d'adaptation de l'ESB, des services déployés, des messages échangés mais aussi de l'infrastructure utilisée dans le but de gérer de façon autonome la QdS et la scalabilité. En

plus des mécanismes de supervision et d'adaptation, cette architecture devra aussi intégrer des stratégies d'analyse des informations issues de la supervision et de planification des actions d'adaptation adéquates à mettre en œuvre.

**Le chapitre 3** présente l'architecture qui décrit la structure et le comportement du bus de service autonome (ASB) que nous proposons.

Des mécanismes « intra-bus » et « extra-bus » ont été conçus pour la gestion de la QdS et la scalabilité de l'ESB. Ces mécanismes sont introduits dans l'architecture de l'ASB et utilisés quand des anomalies sont détectées.

Le cadre de l'autonomic computing proposé par IBM [IBM 05] a été suivi pour avoir une architecture pour la gestion et la coordination des mécanismes proposés.

Une méthodologie de conception guidée par les modèles [SOL 00] a été suivie pour avoir une architecture la plus générique possible. Cette architecture peut donc être instanciée en utilisant toutes les implémentations d'ESB existantes.

Une adaptation de cette architecture en se basant sur le standard JBI [TEN 05] a été faite pour aller vers une implémentation flexible et extensible.

Une plateforme d'émulation a été conçue. Elle est utilisée pour implémenter de façon incrémentale l'architecture proposée. L'implémentation suit une approche basée les différents niveaux de maturité d'un système autonome identifiés par IBM [IBM 05]. En deux chapitres, les différentes étapes pour développer les services et mécanismes permettant de passer du niveau basic au niveau autonome sont présentées.

**Le chapitre 4** détaille les mécanismes et services développés pour avoir une solution manageable basée sur une composant de monitoring développée pour détecter des problèmes de QdS ou de scalabilité de l'ESB. La solution est composée de services qui supervisent l'ESB, les services déployés, les messages échangés et l'infrastructure utilisée, et un module de traitement d'évènements qui agrège, corrèle et filtre les informations issues de la supervision pour la détection d'anomalies.

Des méthodologies pour identifier les anomalies à détecter sont étudiées et proposées.

A ce niveau de maturité, un administrateur de l'ESB l'analyse des anomalies détectées et l'identification du mécanisme adéquat à exécuter pour la correction.

**Le chapitre 5** détaille les propositions pour étendre la solution manageable et aller vers une solution auto-manageable. Ces propositions visent à automatiser l'analyse des anomalies et l'identification du mécanisme adéquat à exécuter pour la correction.

Une première partie du chapitre présente l'approche suivie pour l'analyse des anomalies basée sur un modèle probabiliste qui permet d'inférer l'état global du système en fonction des informations qui caractérisent les anomalies. La connaissance de l'état global du système permet d'identifier la cause des anomalies et de faire un diagnostique.

Une deuxième partie du chapitre présente un modèle sémantique qui guide la définition du plan à exécuter après avoir fait le diagnostique. Ce modèle permet de caractériser les différents mécanismes proposés pour identifier leurs efficacités et leurs limites. Cette caractérisation permet d'identifier devant chaque situation le mécanisme ou la composition de mécanismes à utiliser pour résoudre une anomalie.

Une troisième partie du chapitre présente les services conçus pour la coordination et l'exécution des actions nécessaires pour la mise en place du plan défini.

**Le chapitre 6** présente le projet européen IMAGINE qui a guidé les scénarios de validation des services et des mécanismes proposés pour l'ASB.

La plateforme d'émulation proposé pour l'implémentation de l'ASB a été utilisé pour évaluer nos différentes contributions. Les résultats montrent l'impact négligeable des services de supervision, l'amélioration induite par l'application des mécanismes « intra-bus » ou « extra-bus » proposés. La plateforme d'émulation a aussi permis de collecter des données permettant de construire le modèle probabiliste permettant d'inférer l'état global du système en fonction des anomalies, mais aussi d'instancier le modèle qui contient les caractérisations des différents mécanismes « intra-bus » et « extra-bus ».

Ces résultats valident la proposition de l'ASB comme une solution capable de garantir de façon autonome la QdS et la scalabilité pour les systèmes distribués.

**Le chapitre 7** donne une conclusion globale du travail avec les différentes perspectives qui ont été identifiées.

Les perspectives de recherche qui poursuivront ce travail de thèse sont :

- l'extension de l'architecture proposée pour inclure les autres middleware qui permettront de mieux garantir la QdS et la scalabilité ;
- l'extension de la solution de monitoring proposée pour aller vers un monitoring dynamique et intelligent ;
- l'extension du modèle utilisé pour le plan en rajouter à la caractérisation des mécanismes les processus d'exécution associés à chaque mécanisme ;
- la proposition de l'ASB comme une solution Cloud de niveau PaaS ;
- la proposition d'un autonomic manager comme une solution Cloud de niveau SaaS.

# Bibliography

[ALV 12] Alvaro, V., and Williams, Jason J. W., “RabbitMQ in Action Distributed Messaging for Everyone”, Shelter Island, NY: Manning, 2012. ISBN 1935182978, 9781935182979

[AMQ 12] OASIS Advanced Message Queuing Protocol, Version 1.0, OASIS standard, October 12, available at http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

[BAR 12] Barber, D., “Bayesian Reasoning and Machine Learning”, New York, NY, USA: Cambridge University Press, 2012.

[BIG 02] Bigus, J. P., Schlosnagle, D. A., Pilgrim, J. R., III, W. N. M., and Diao, Y., "ABLE: A toolkit for building multiagent autonomic systems", IBM Systems Journal 41, 3, 350–371. 2002.

[BIR 84] Birrell, A. D. and Nelson, B. J., "Implementing remote procedure calls", ACM Transactions on Computer Systems 2(1), pp. 39-59, March 1984.

[BIR 02] Birman, K.P.; van Renesse, R.; Vogels, W.; "Scalable data fusion using Astrolabe," Information Fusion, 2002. Proceedings of the Fifth International Conference on , vol.2, no., pp. 1434- 1441 vol.2, 2002

[BRA 94] Braden, R., Clark, D., Shenker, S., Integrated Services in the Internet Architecture: an Overview, RFC 1633, June, 1994.

[BRA 97] Braden, R., Zhang, Ed. L., Berson, S., Herzog, S., Jamin, S., Resource ReSerVation Protocol (RSVP), RFC 2205, September, 1997.

[BUC 04] Buchmann, A., Bornovd, C., Cilia, M., Feige, L., et al, “DREAM : Distributed Reliable Event-based Application Management,” P. A Levene M, Ed. : Springer-Verlag, 2004, pp. 319-352

[CAL 08] Callaway, R.D., Viniotis, Y., Rodriguez, A., Brown, K.G. and Robinson R., “Enabling federations of enterprise service buses using a distributed service registry”. Proceedings of the Second international workshop and Summer School on Service Science, Management and Engineering, SSME2008, Palermo Italy, 2-6 June 2008.

[CAL 09] Calinescu, R. and Kwiatkowska, M. 2009. "Using Quantitative Analysis To Implement Autonomic IT Systems", in Proceedings of the IEEE 31st international Conference on Software Engineering, Vancouver, Canada, May 16-24, 2009

[CAR 02] Carpenter, B. and Brim, S., “Middleboxes: Taxonomy and Issues”, RFC 3234, February 2002.

[CHA 04] Chappell, D. A., “Enterprise Service Bus”, O’Reilly Media, June 2004.

[CHA 10] Chanda J., Sengupta S., Kanjilal A., Bhattacharya S., “CA-Graph: A Context Aware Graph to model Enterprise Service Bus”, IEEE India Conference (INDICON), 1 - 4 2010, Kolkata

[CHA 11] Chanda J., Sengupta S., Kanjilal A. and Bhattacharya S., “CA-ESB: Context Aware Enterprise Service Bus”, International Journal of Computer Applications 30(3): 1-8, September 2011. Published by Foundation of Computer Science, New York, USA

[CHE 06] Chen Q., Yao J., Xing R., “Middleware Components for E-commerce Infrastructure: An Analytical Review”, Informing Science and Information Technology, Volume 3, 2006

[CIL 12] Cilia mediation framework, available at http://wikiadele.imag.fr/index.php/Cilia, 2012

[COM 13] COMPONENT OBJECT MODEL TECHNOLOGIES, 2013. Available at http://www.microsoft.com/com.

[COR 13] CORBA OMG, 2013. Available at http://www.corba.org.

[CUR 04] Curry, E., "Message-Oriented Middleware", in Middleware for Communications, Q. H. Mahmoud, Ed. Chichester, England: John Wiley and Sons, 2004, pp. 1-28.

[DAS 02] Dashofy, E. M., van der Hoek, A., and Taylor, R. N., "Towards architecture-based self-healing systems", first workshop on Self-healing systems, 2002.

[DEV 06] Devitt, A., Danev, B., and Matusikova, K. 2006. “Constructing bayesian networks automatically using ontologies”, 2nd Workshop on Formal Ontologies Meets Industry (FOMI 2006).

[DIO 13] Diop, C., Mezghani, E., Exposito, E., Chassot, C., Drira, K., “QoS-driven autonomic abilities through a multi-homed transport protocol”, IEEE International Conference on Advanced Information Networking and Applications (IEEE AINA), Barcelona, Spain, mars, 2013, 8p.

[DZU 13] DZUL, R. K., Vargas-Santiago, M., Diop, C., Exposito, E., Moo-Mena, F., “A smart diagnostic model for an autonomic service bus based on a probabilistic reasoning approach”, 10th IEEE International Conference on Autonomic and Trusted Computing (UIC/ATC 2013), Vietri sul Mare, Italia, 18-21 December 2013, pp.416-421.

[EEL 05] Eeles, P., Capturing Architectural Requirements, November 15th 2005, Available at http://www.ibm.com/developerworks/rational/library/4706.html#N100A7

[EMM 00] Emmerich, W., "Software engineering and middleware: a roadmap", Conference on The future of Software engineering , pp. 117-129, 2000.

[ESP 06] Esper reference 4.6.0 by Esper Team and EsperTech Inc.[http://esper.codehaus.org] 2006. Available at http://esper.codehaus.org/esper-4.6.0/doc/reference/en-US/pdf/esper_reference.pdf

[ETZ 10] ETZION O., NIBLETT P., “Event Processing in Action”, p. 384, 12 August 2010.

[EXP 12] Exposito, E. “Advanced Transport Protocols: Designing the Next Generation”, Wiley-ISTE, 2012.

[EXP 14] Exposito, E., Diop, C., “Smart SOA platforms in cloud computing architectures”, Wiley-ISTE, ISBN: 978-1-84821-584-9, June 2014, pp.224.

[FIE 00] Fielding, Roy T., “Architectural Styles and the design of Networked Based Software architectures”, PHD Thesis, University of California, Irvine 2000.

[FOR 10] A. Ford, C. Raiciu, S. Barre, J. Iyengar, “Architectural Guidelines for Multipath TCP Development”, IETF Internet-draft, June, 2010.

[FOR 11] A. Ford, C. Raiciu, M. Handley et al, “TCP Extensions for Multipath Operation with Multiple Addresses”, Work in Progress, March 2011.

[FOU 08] Fournier-Morel X., Grojean, P., Plouin, G., Rognon, C., "SOA– Le guide de l'architecte d'un SI agile: Le guide de l'architecte d'un SI agile », 3ème édition, Dunod, 2011, ISBN 2100568523, 9782100568529, Dunod

[FUR 11] Furdik, K., Sabol, T., Hreno, J., Bednar, P., Lukac, G. and Mach, M., “A Platform for Semantically Enhanced Business Collaboration of Networked Enterprises”, Introduction to the Semantic Web: Concepts, Technologies and Applications, iConcept Press Ltd., Hong Kong, China, 2011, pp. 121-144.

[GAC 14] GARTNER, “Gartner IT glossary”, January 2014. Available at http://www.gartner.com/it-glossary/cloud-computing/.

[GAR 02] Garlan, D., Schmerl, B., Cheng, S.-W., Sousa, J. P., Spitznagel, B., and Steenkiste, P., "Using architectural style as a basis for system self-repair". In Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture, 2002.

[GAR 13] GARTNER, “Gartner IT glossary”, December 2013. Available at http://www.gartner.com/it-glossary/eda-event-driven-architecture.

[GON 11] Gonzalez, L., Ruggia, R., “Addressing QoS Issues in service Based Systems through an Adaptive ESB Infrastructure”, ACM MW4SOC, 2011, December 12th, Lisbon Portugal.

[GON 13] González L., Laborde J. L., Galnares, M., Fenoglio, M., Ruggia, R., “An Adaptive Enterprise Service Bus Infrastructure for Service Based Systems”. ICSOC Workshops 2013: 480-491

[GRU 95] Gruber, T. R., “Towards principles for the design of ontologies used for knowledge sharing”, Int. J. Hum. Comput. Stud. 43, 5-6 Dec. 1995, 907-928.

[HOH 03] Hohpe, G., Woolf, B., “Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions”, ISBN-13: 078-5342200683, Addison-Wesley Professional; 1 edition (October 20, 2003)

[IBM 05] IBM. An Architectural Blueprint for Autonomic Computing. IBM White Paper 3th Ed., June 2005.

[IFT 12] Iftikhar, U. and Weyns, D., "A Case Study on Formal Verification of Self-Adaptive Behaviors in a Decentralized System", 11th International Workshop on Foundations of Coordination Languages and Self Adaptation (FOCLASA'12) EPTCS 91, 2012, pp. 45–62

[IHR 13] IHRIS, iHRIS open source software – health sector, December 2013. Available at http://www.ihris.org/wiki/Non-functional_Requirements.

[ILG 13] D. Gil de la Iglesia and Weyns D., "Guaranteeing robustness in a mobile learning application using formally verified MAPE loops". International Symposium on Software Engineering for Adaptive and Self-Managing Systems. 2013, p83–92

[IMA 11] Innovative End-to-end Management IMAGINE Project, http://www.imagine-futurefactory.eu/index.dlg

[ISS 11] Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadis, P., Autili, M., Gerosa, M. A. and Hamida, A. B. "Service-Oriented Middleware for the Future Internet: State of the Art and Research Directions", in: Journal of Internet Services and Applications (JISA)(1-23) 2011

[ITU 97] ITU-T Information Technology – Quality of Service Framework, ITU-T X.641 (ISO/IEC IS13236) Dec 1997.

[ITU 08] UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification, 1.1, OMG, April 2008

[JCO 14] JConsole: the Java Monitoring and Management Console. Available at http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html (last access July 2014)

[JOL 14] Jolokia: an HTTP/JSON bridge for remote JMX access. Available at http://www.jolokia.org/ (last access July 2014)

[KAR 07] Karastoyanova, D.; Wetzstein, B.; van Lessen, T.; Wutke, D.; Nitzsche, J.; Leymann, F.; "Semantic Service Bus: Architecture and Implementation of a Next Generation Middleward", Data Engineering Workshop, 23rd International Conference on Data Engineering, ICDE 2007, April 15-20, 2007, Istanbul, Turkey.

[KAZ 09] Kazhamiakin, R. "Adaptation and Monitoring in S-Cube: Global Vision and Roadmap," Proceedings of the Workshop on Monitoring, Adaptation and Beyond (MONA+), Madrid, Spain. (June 2009): 67-76.

[KEP 03] KEPHART J.O., CHESS D.M., "The vision of autonomic computing", Computer, vol. 36, no. 1, pp. 41–50, January 2003.

[KOH 06] Kohler, E., Handley, M., and Floyd, S., Datagram Congestion Control Protocol, IETF, RFC 4340, March 2006. E. Kohler, M. Handley, and S. Floyd, Datagram Congestion Control Protocol, IETF, RFC 4340, March 2006.

[KRA 04] KRAFZIG D., BANKE K., SLAMA D., "Enterprise SOA: Service-Oriented Architecture Best Practices", Prentice Hall, 2004.

[KRI 07] Kritikos, K., Plexousakis D., "OWL-Q for semantic QoS-based Web Service description and Discovery", 1rst workshop on service Matchmatching and Resource Retrieval in the semantic Web, SMRR, 2007.

[KWI 05] Kwiatkowska M., "Quantitative analysis with the probabilistic model checker PRISM." Electronic Notes in Theoretical Computer Science, 153(2):5–31, 2005.

[LIN 13] Lindholm, T., Yellin, F., Bracha, G., Buckley, A., The Java Virtual Machine Specification Java SE 7 Edition, 2013. Available at http://docs.oracle.com/javase/specs/jvms/se7/html/

[LIU 08] Liu, Y., Gorton, I., Clayphan, A. J., "An Autonomic Middleware Solution for Coordinating Multiple QoS Controls", ICSOC '08 Proceedings of the 6th International Conference on Service-Oriented Computing, Pages 225 – 240.

[MAR 09] March, M., Bednar, P., Furdik, K., "Support for Forming Temporal Business Alliances as Networked Enterprises," Proc. of CECIIS 2009, Varaždin, Croatia, University of Zagreb, Faculty of Organisation and Informatics, 2009, 179-186.

[MAS 10] Masternak, T., Psuik, M., Radziszowski, D., Szydlo, T., Szymacha, R., Zielinski, K., and Zmuda, D., "ESB-Modern SOA Infrastructure", SOA Infrastructure Tools, Concepts And Methods, Poznan University of Economics Press, Dec. 2010

[MIC 06] Michelson, B., "Event-driven architecture overview," Patricia Seybold Group, Feb, 2006

[MIE 09] Mietzner, R., Van Lessen, T. ; Wiese, A. ; Wieland, M. ; Karastoyanova, D. ; Leymann, F., “Virtualizing Services and Resources with ProBus: The WS-Policy-Aware Service and Resource Bus”, IEEE International Conference on Web Services, 2009. ICWS 2009. 617-624, Los Angeles, July 2009

[MIS 13] MICROSOFT, .NET Remoting: Core Protocol, December 2013. Available at http://msdn.microsoft.com/en-us/library/cc237297.aspx.

[MOR 13] Morand, D., "Cilia: un framework pour le développement d’applications de médiation autonomiques", PhD thesis, 2013. Available at http://tel.archives-ouvertes.fr/docs/00/95/21/07/PDF/morand-thesis-final.pdf

[NAG 14] Nagios: The Industry Standard In IT Infrastructure Monitoring, Available at http://www.nagios.org/ (last access July 2014)

[NIC 98] Nichols, K., Blake, S., Baker, F., Black, D., Definition of the Differentiated Services Field (DS Field), RFC 2474, December, 1998.

[NOR 13] NFP OPEN REQUIREMENTS PROJECT, Collaborative requirements for the non-profit sector, December 2013. Available at http://www.nfprequirements.org

[ODA 06] Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering, W3C working draft, 2006.

[OMG 04] OMG Data-Distribution Service for Real-Time Systems, Version 1.0, December 2004 http://www.omg.org/spec/DDS/1.0/

[OMG 08] UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification, 1.1, OMG, April 2008

[OPS 14] Opsview: a powerful monitoring for IT systems. Available at http://www.opsview.com/ (last access July 2014)

[ORT 07] Ortiz, S., "Getting on board the enterprise service bus, " Computer, vol. 40, no. 4, pp. 15-17, April.

[OSS 01] Ossama O., Carlos O., and Douglas C. S., “The Design and Performance of an Adaptive CORBA Load Balancing Service”, Distributed Systems Engineering Journal’s “online” edition. February 3, 2001

[OWL 04] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, P., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K., “OWL-S Semantic Markup for Web Services”, W3C submission 2004, http://www.w3.org/Submission/OWL-S/

[PAN 12] Pan, G., Zhang, L., Wu, Z., Li, S., Yang, L., Lin, M., Shi, Y., “Pervasive Service Bus: Smart SOA Infrastructure for Ambient Intelligence”, Intelligent Systems IEEE, 10.1109/MIS.2012.119, December 2012

[PEN 09] Pencolé, Y., Subias, A., “A Chronicle-based Diagnosability Approach for Discrete Timed-event Systems: Application to Web-Services”, J. UCS 15(17): 3246-3272 (2009)

[PEZ 11] Pezzini, M., BLheureux B., Integration Platform As A Service: Moving integration to the Cloud, Gartner RAS Core Research Note GOO210747, 7 March 2011

[PIE 03] Pietzuch, P. R. and Bhola., S., “Congestion control in a reliable scalable message-oriented middleware”, In Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware (Middleware '03), Markus Endler (Ed.). Springer-Verlag New York, Inc., New York, NY, USA, 202-221.

[POS 81] Postel, J., Transmission Control Protocol; DARPA Internet Program Protocol Specification, RFC 793, September 1981.

[POS 80] Postel, J., User Datagram Protocol, RFC 768, August 1980.

[PRO 14] Proxmox Virtual Environment: a complete server virtualization management solution http://www.proxmox.com/ (last access July 2014)

[PSI 12] Psiuk, M., Bujok, T., Zielinski, K., “Enterprise Service Bus Monitoring Framework for SOA Systems”. IEEE T. Services Computing 5(3): 450-466 (2012)

[RMI 13] REMOTE METHOD INVOCATION SPECIFICATION, December 2013. Available at http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html.

[ROM 05] Roman, D., Keller, U., Lausen, H., Jos de Bruijn, Lara, R., Stollberg, M., Polleres, A., Feier, C., Christoph Bussler, and Dieter Fensel: Web Service Modeling Ontology, Applied Ontology, 1(1): 77 - 106, 2005.

[RRD 12] RRDtool: an OpenSource industry standard, high performance data logging and graphing system for time series data, 2012. Available http://oss.oetiker.ch/rrdtool/index.en.html

[SIL 11] Silcher, S., Minguez, J., Mitschang, B., "Adopting the Manufacturing Service Bus in a Service-based Product Lifecycle Management Architecture". In: Proceedings of the 44th International CIRP Conference on Manufacturing Systems: ICMS '11; Madison, Wisconsin, USA, May 31 - June 3, 2011.

[SOA 06] SOA RM, Reference Model for Service Oriented Architecture 1.0, OASIS Standard, October 2006.

[SOL 00] Soley, R. and the OMG Staff Strategy Group, Object Management Group, White Paper, Draft 3.2 – November 27, 2000

[SPI 95] Spirtes, P., and Meek, C., "Learning bayesian networks with discrete variables from data," in Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95), Montreal, Canada, August 20-21, 1995, pp. 294–299, AAAI Press, 1995.

[SPI 01] Spirtes, P., Glymour, C., and Scheines, R., "Causation, Prediction, and Search." Cambridge, MA, USA: The MIT Press, second ed., Jan. 2001.

[STE 05] Sterritt, R., Parashar, M., Tianfield, H., and Unland, R., "A concise introduction to autonomic computing." In Advanced Engineering Informatics, volume 19, pages 181–187, January 2005.

[STE 07] Stewart, R., Stream Control Transmission Protocol, IETF RFC 4960, Sept. 2007.

[TEN 05] Ten-Hove, R., Walker, P., JBI JSR-000208 Java Business Integration 1.0, Public Review Draft March 2, 2005, Available at http://jcp.org/aboutJava/communityprocess/final/jsr208.

[THE 03] Thelin, J., "A Comparison of Service-oriented,Resource-oriented, and Object-oriented Architecture Styles", 2003 Cape Clear Software Inc.

[TON 08] Tondello G. F., Siqueira F., "The QoS-Mo ontology for semantic QoS modelling", ACM symposium on Applied Computing 2008, New York.

[UDD 04] Universal Description Discovery and Integration (UDDI) Version 3.0.2, OASIS Technical Specification, 19 October 2004, Available at http://www.uddi.org/pubs/uddi-v3.0.2-20041019.htm

[VEN 98] Venners, B., "Inside the Java Virtual Machine (chapter 5 The Java Virtual Machine)", pp.703 ISBN 0-07-135093-4, Computing McGraw-Hill 1998. Available at http://www.artima.com/insidejvm/ed2/index.html

[VIS 14] VisualVM: a tool to monitor and troubleshoot Java applications Available at http://visualvm.java.net/ (last access July 2014)

[VIZ 13] Vizcarrondo, J., Aguilar, J., Subias, A., Exposito, E.: "Distributed chronicles for recognition of failures in web services composition." CLEI 2013: 1-10

[WAM 09] Wambeke N. V., "Une approche pour la composition autonome de services de communication orientés QoS : Application aux protocoles de transport configurables", 2009

[WAM 13] Wambeke N. V., Ernesto Exposito, Christophe Chassot, Michel Diaz, "ATP: A Microprotocol Approach to Autonomic Communication", IEEE Trans. Computers 62(11): 2131-2140 (2013)

[WOL 07] John, Wolfgang & Tafvelin, Sven, "Analysis of Internet Backbone Traffic and Header Anomalies Observed. " IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, Pp 111-116. October 2007.

[WOO 95] Wooldridge, M. and Jennings, N. 1995. "Intelligent agents: Theory and practice." Knowledge Engineering Review 10, 2, 115–152.

[WSD 01] "Web Service Definition Language (WSDL)," 1.1, W3C NOTE 15 March 2001. http://www.w3.org/TR/wsdl

[WSD 07] Web Service Description Language, "Web Service Definition Language (WSDL)," 2.0, W3C Recommandation, 26 June 2007, http://www.w3.org/TR/wsdl20-primer/

[WSO 11] WSO2 Case Study, eBay uses 100% open source WSO2 ESB to process more than 1 billion transactions per day, http://wso2.com/casestudies/ebay-uses-100-open-source-wso2-esb-to-process-more-than-1-billion-transactions-per-day/

[WSP 07] W3C. Web Services Policy 1.5 - Framework, W3C Recommendation, 2007.

[WU 09] Wu, B., Liu, S., Cui, W., “Double Redundant Fault-Tolerance Service Routing Model in ESB”, 4th ChinaGrid Annual Conference, 21-22 August 2009.

[WU 10] Wu, Q., Li, P., “Study and Implement of Dynamic Routing Based on QoS in Enterprise Service Bus”, Journal of Computation Information Systems, 6:7, pp 2093-2098, July, 2010.

[YAN 04] Yang, M., Huang, Y., Kim, J., Lee, M., Suda, T., Daisuke, M., “An End-to-End QoS Framework with On-Demand Bandwidth Reconfiguration”, IEEE INFOCOM 2004

[YUA 12] Yuan, C., Dan, N., Hua-Ji, S., Jia-Hu, G., “Research on dynamic message routing for ESB based on message system”, International Conference on Computer Science and Service System, 2012, pp 663-665.

[ZHO 04] Zhou, C.,Chia L.T., Lee B. S. “DAML-QoS ontology for Web services”, IEEE International Conference on web Services 2004, Washington, DC.

[ZHO 10] Zhou, C., Zhang, X., “A Policy-Configurable Dynamic Routing Mechanism in Enterprise Service Bus”, International Conference on Educational an Information Technology, September, 2010, Chogqing, China.