POLITECNICO DI MILANO

Corso di Laurea Specialiatica in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

A Framework for Managing the INDENICA Virtual Service Platform

Relatore: Prof. Dr. Luciano Baresi
co-Relatori: Dr. Benjamin Satzger
Prof. Dr. Schahram Dustdar
Prof. Dr. Sam Jesus Guinea Montalvo

Tesi di Laurea Specialistica di:
Conte Luigi - Mat. 750214

Anno Accademico 2010-2011

Ringraziamenti

Giunto al termine di questo lavoro desidero ringraziare ed esprimere la mia riconoscenza nei confronti di tutte le persone che, in modi diversi, mi sono state vicine e hanno permesso e incoraggiato sia i miei studi che la realizzazione e stesura di questa tesi.
Un grazie di cuore alla mia famiglia che mi ha aiutato in ogni modo in tutti questi anni di studio, senza di loro non avrei mai raggiunto questa meta!

I miei più sentiti ringraziamenti vanno a chi mi ha seguito durante la redazione del lavoro di tesi, il *Prof. Benjamin Satzger*, dandomi consigli e confronti che mi hanno aiutato ad intraprendere, ogni volta, le scelte più appropriate. Come non dimenticare il *Dr. Prof. Luciano Baresi* il *Dr. Prof. Sam Guinea* che non solo mi hanno permesso di fare questa esperienza presso la *Technische Universität Wien* ma sono stati sempre disponibili a risolvere qualsiasi mia richiesta di aiuto. Un ringraziamento particolare al *Dr. Prof. Shahram Dustdar* per la fiducia fin da subito dimostratami nell'assegnarmi questo argomento di tesi e per non avermi fatto mancare nulla durante la mia permanenza a Vienna. Devo inoltre ringraziare per la loro disponibilità *Christian Inzinger*, *Philipp Leitner* e *Waldemar Hammer* che mi hanno for-

nito dei suggerimenti più che utili alla conclusione del mio lavoro di tesi.

Come non ringraziare tutti gli amici del *PoliMi*, gli amici del gruppo sportivo *CUS Milano* ed in modo particolare *DiGio* e *Jhon* (sì, la "h" va proprio lì) con i quali ho affrontato questi anni di intenso studio, seguito numerosi corsi e sono sempre stati accanto a me anche durante la mia permanenza all'estero.

Voglio ringraziare tutti i miei amici, che in questi anni non mi hanno mai fatto mancare affetto e fiducia e mi sono stati accanto nei momenti più difficili. Il loro prezioso aiuto e i loro consigli sono stati fondamentali anche per il raggiungimento di questo traguardo. Un ringraziamento particolare a *Giorgio*, *Silvia* e a tutti gli amici del gruppo *exchange students in Vienna* che spesso mi hanno permesso di staccarmi dal mio lavoro di tesi per qualche ora di sano divertimento ⌣̈.

Abstract

Today, companies have to cope with an increasingly heterogeneous IT landscape and it is getting more and more challenging to integrate and govern their enterprise information systems. Self-adaptive systems attracted significant attention but we need a good software technology that enables dynamic and reconfigurable software development. A lot of development approaches were studied and used in order to develop these self-adaptive systems. Service-oriented architectures (SOA) have been identified as a partial solution to these problems and have been widely adopted. This has led to a high number of service platforms that are available today. To shield applications from the heterogeneity of the different service platforms the logical next step is to create virtual service platforms (VSP). Such VPSs seem promising but as of today there are no approaches for how to manage them. In this work we propose a framework that helps to manage and control VSPs. The framework combines ideas from the areas of Autonomic Computing and adaptive systems. It's main components are a monitoring engine based on complex event processing and an adaptation engine based on a business rule engine. The engines can be flexibly structured and combined to adjust to the infrastructure that is to be managed. Moreover, graphical user interfaces are provided that will enable the clients to define rules and to deploy them into the VSP. We conduct a case study based on a warehouse management that will be able to demonstrate the functionality and performance of the framework.

Estratto

Negli ultimi anni molte aziende hanno sviluppato diverse soluzioni di sistemi informativi nella propria infrastruttura. A causa della diversa natura di questi sistemi informativi, queste aziende hanno avuto il problema di dover integrare e amministrare questi sistemi eterogenei. Pensare di voler gestire insieme questi sistemi con la sola forza umana è ormai impossibile e i confini di questi sistemi non sono più interni alla sola azienda. Una possibile soluzione alla loro gestione, che ha inoltre suscitato particolare attenzione, sono i sistemi auto-adattativi. Abbiamo però bisogno di una tecnologia software adatta allo sviluppo di software dinamico e riconfigurabile. Sono stati studiati diversi approcci di sviluppo software per questa tipologia di sistemi auto-adattativi e l'architettura orientata ai servizi (SOA) sembra essere una buona soluzione adatta a migliorare la loro accuratezza, il tutto portando alla soddisfazione dei requisiti di business. Infatti, una composizione di servizi può essere riorganizzata dinamicamente senza il bisogno di spegnere i singoli servizi ma soprattutto in modo trasparente all'utente. Questo è il motivo per cui abbiamo bisogno di una infrastruttura che possa elaborare decisioni dinamicamente e a runtime. Un altro aspetto da non sottovalutare è che vi è una crescente dipendenza da servizi esterni oltre che una crescente frammentazione di questo ecosistema di servizi e lo sviluppatore di servizi si troverà a doverli orchestrare. La creazione di piattaforme virtuali di servizi (VSP) da parte delle aziende sarà un modo per evitare possibili influenze nelle applicazioni della propria infrastruttura. Le VSP sono una buona soluzione e creano un livello di astrazione sui propri servizi ma nella situazione attuale non sappiamo ancora come realmente amministrarle. Durante questi ultimi anni sono stati condotti diversi studi di ricerca su quale possa essere il miglior design per le VSP. In aggiunta, numerosi studi di ricerca si sono focalizzati sulla modellizzazione di sistemi auto-adattativi che possano abilitare la gestione delle VSP ed alcuni hanno portato anche allo sviluppo e implementazione di alcuni prototipi.
Attenzione particolare in questo lavoro di tesi è stata rivolta nell'identificazio-

ne di un modello di gestione di VSP basato su sistemi auto-adattativi ed autonomic-computing e che allo stesso tempo potesse ereditare le migliori caratteristiche degli studi precedenti in modo da amministrare al meglio questi sistemi eterogenei. Introdurremo diverse tecnologie soffermandoci in modo particolare su componenti per l'elaborazione di eventi complessi (CEP) che consentiranno di filtrare e aggregare eventi provenienti dai servizi monitorati nella VSP. Verrà definito anche un modello di eventi della VSP e che i diversi servizi monitorati dovranno rispettare qualora vorranno mandare informazioni al framework di gestione della VSP. Dopo aver implementato la parte di monitoraggio ed elaborazione delle informazioni del framework, avremo bisogno di un componente in grado di definire ed integrare nella VSP politiche di business. Infine, verranno sviluppati dei componenti in grado di applicare adattamenti sui sistemi monitorati. Sarà anche studiato un meccanismo di comunicazione tra i componenti interni al framework adatto a supportare la flessibilità degli stessi componenti voluta. Questo consentirà di avere diversi livelli gerarchici di monitoraggio e adattamento in modo da definire politiche di basso livello per i singoli sistemi monitorati e politiche di alto livello per la completa gestione dinamica della stessa VSP. In aggiunta, saranno sviluppati un servizio adatto alla memorizzazione delle informazioni che il framework necessita (ad esempio, le diverse regole di CEP e politiche di business saranno salvate in questo componente) e un tool completamente grafico che consentirà ai clienti di definire le diverse regole di monitoraggio e adattamento, i parametri di configurazione di sistema e il binding tra eventi tipici del sistema monitorato con il modello di eventi definito nel framework oltre che salvarle nello stesso.
Dopo aver disegnato e implementato questo framework, forniremo un caso di studio basato sulla gestione di una warehouse. Questo caso di studio ci permetterà di testare l'effettivo funzionamento della piattaforma oltre che mostrare le prestazioni.

Contents

List of Figures

Chapter 1

Introduction

The following chapter shows the general scenario in which we had to work. In particular, in Section 1.1 *we try to describe the general problem in the actual information systems. In* Section 1.2 *we give a motivation of the problems introduced before. In* Section 1.3 *we describe our contribution in order to solve the problems introduced. Finally, in* Section 1.4 *we introduce and explain all the contents in the next chapters.*

1.1 Analysis of the problem

Nowadays, in particular for companies that implemented different information systems solutions during the years, there are many problems connected to the actual information systems nature. For example, implementing different solutions during the years leads to the huge problem of integrating these solutions in order to let them work together. As a result, these companies are affected by the managing problem of these heterogeneous services. Trying to manage everything by considering it as a closed world environment is unrealistic. In fact dynamic and open environments are now the norm and many unpredictable stakeholders arise. For these reasons requirements cannot be detected in advance thus we need a flexible support for the changes. This has led to the use of many development approaches like incremental, prototype-based or modular because changes need redeployment. Incremental (or iterative) development is a cyclic software development process created in order to solve the problems that the waterfall model had. Like the waterfall model, it starts by planning and finishes with the deployment phase, but there could be many cycles in the development process before reaching the exit (deployment). In this way the developer can fix the deployed system

because, for example, some tests failed or some requirements were not satisfied. This approach has some disadvantages like rigidity for each phase (not overlap each other) and mapping requirements to increments may be not easy. Prototype-based development is an object oriented approach in which there are not classes. The main feature of this development approach is the *delegation*: the language runtime dispatches the right method or finds the right piece of data by simply following a series of delegation pointers until it finds a match. The major disadvantages of prototype-based development are that the system structure can be damaged in case of many changes and that it doesn't fit on large applications. Modular development uses explicit models that describe development activities and products. This means that diagrams are used instead of the code for the development. By making process and product models, the developer is allowed to define and use complex steps during the development that are correct by design. The main disadvantage for this development approach is the time to draw and validate the models [6].
Another development approach that had a good success within the last years in many application domains is the component-based development. A software component is a software package, a Web service or a module that encapsulates a set of functions or data. Examples in which component-based approach is successfully used are web-based systems, desktop and graphical applications. The communication between these components is obtained by using the interfaces. An important feature of this approach is that is possible to substitute components also at runtime by just taking care of using the same interface [14].
A component-based software is actually one of the best solutions to the problems mentioned before. In this scenario, services become key actors in open-world systems, thus resources are available on a network as services [3]. The interface and programming model for application service development in a service platform is made up by some infrastructure assets (e.g., communication middleware or databases) and platform services. In a service platform we have variance of functional and non-functional properties, moreover its interface varies with the requirements of the domain. For example, the mass data storage and scalability are important for enterprise systems security services but they are engineering-related services in the industry automation domain. Integrated service applications in the Internet of the Future will need platforms designed based on both the specific functional and non-functional needs, and support to services integration across platform boundaries because service-based applications will integrate services from multiple platforms from different domains.

By taking a look at the companies, their matured experience tells that:

- different domains (like embedded real time vs. telecom real time vs. web-based information systems) need different service platforms (sometimes still have to be developed) tailored to fit that specific domain;
- the Internet of the Future will lead to an integration over those tailored service platforms;
- the already fragmented service platforms prevent the integration process of services between platforms.

In order to have the possibility to use combinations of hardware and software that could best fit a part of the enterprise [54], we need high heterogeneity. This high heterogeneity will also help us to meet all the different functional and non-functional requirements. In a scenario with high heterogeneity, the same data could be represented in different ways but most of that data can be marshalled from one system to another one or some components in the distributed system can have different capabilities than other ones [8]. There is a common trend to develop heterogeneous systems because most of the organisations run applications on different platforms that might also support different languages. Heterogeneous systems, like all software systems, need to evolve when requirements change. Unfortunately, supporting the evolution of these systems is very difficult because of the different platforms and different languages used [18].

1.2 Motivation

As far as we are in a scenario characterised by several heterogeneous systems and we want to use them together, a good idea is to do this by using a virtual service platform that could provide a globally integrated environment of these heterogeneous services in which they can cooperate. The service-centric computing vision will help us to get all these services cooperate and form a service ecosystem.
A lot of service platforms for use by applications are already available on the market and in the future there will be more of these platforms. There is also an increasing need to merge non-SOA platforms into service-based applications. These are some of the reasons because is needed high degree and heterogeneity of the already existing platforms instead of less heterogeneity. The result of the missing heterogeneity is different types of fragmentation of service platforms:

Quality fragmentation : several service platforms are needed in order to support all the Quality of Service (QoS) requirements like latency, availability, throughput, reliability, etc.

Interface fragmentation : usually interfaces for services are different even if they offer the same service behaviour so a binding to a specific interface results in a dependency on that service platform supplier.

Technical fragmentation : there are different situations when we may have this fragmentation but are all related to the different technologies used to implement the same service.

These three forms of fragmentation slow the development of service-based applications and so the growing up of a service ecosystem, in particular large-scale integrated applications crossing several domain borders. Moreover, the fragmentation of these complex distributed heterogeneous systems makes the control very difficult for the companies so we need a new development approach for these service platforms.
Actually, the challenge is not only administering the whole complex distributed environment. Regarding the service platform, the platform service developer can be both client and consumer of the service platform's services and he furnishes the domain-specific business logic. The application developer has the role of orchestrating these services by having in mind the domain-specific business logic. All these factors lead to an increase in business service's dependency on the functioning and availability of platform services.
In order to survive, in particular with the growing of services eco-systems, the service provider has to consequently minimise the effects of external and internal (in-house) influence and has to invest into application development. The growing dependency on external services and the fragmentation will lead the companies to create Virtual Service Platforms in order to prevent those influences in their application infrastructure. But we will also need advanced interoperability and portability constraints in case of domains where appropriate platforms are still missing nowadays. These Virtual Service Platforms can vary based on the application domain but the general form is obtained by creating an abstraction layer over the existing services.
INDENICA[30] is a research project co-funded by the European Commission within the 7th Framework Programme in the area Internet of Services, Software & Virtualisation (ICT-2009.1.2) and it tries to discover the best practices to build these Virtual Service Platforms by providing an exemplary implementation. An outline of the Virtual Service Platform role in

a heterogeneous service environment and the three kinds of fragmentation described before can be found in Figure 1.1. That schema is a good example

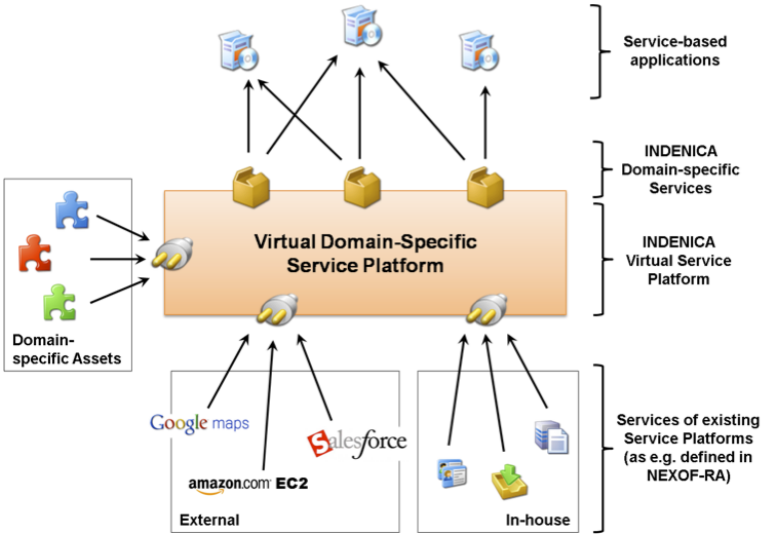

Figure 1.1: Virtual Service Platform in a heterogeneous service environment.

for identifying different kinds of fragmentation:

- technical fragmentation because of the multiple technologies, middleware, protocols (mainly contained in the layer of external services and in-house services that are used in the Virtual Service Platforms);
- These service-based interfaces suffer from the interface fragmentation;
- We also face quality fragmentation when domain-specific assets are spread around the different layers of the service-oriented architecture.

Virtual domain-specific service platforms can be used for several purposes but the most important from our point of view are:

- the access to application-specific services is centralised by the integration of external service platforms;
- offering protection of service and application infrastructures against potential discontinuation of external or internal services;
- integrating application and/or domain-specific functionality with in-house and external services;
- supporting vertical and horizontal integration of services over domain and abstraction level borders;

- providing integrated system management functionality and support of QoS monitoring and flexible adaptation and configuration mechanisms.

Services and applications are then enabled by the Virtual Service Platforms to view the all service environment as a complete defragged service delivery platform independent from the technical realisation. We have a domain-specific view of external and internal services by using the Virtual Service Platform. Therefore the growing dependency on external service and platform vendors can be largely avoided and will make the real-time enterprise a reality by simplifying the development of the applications. The use of these Virtual Service Platforms with all their ramifications to enable the support for horizontal and vertical service integration goes beyond the enterprise level being an important part of the service and service platform ecosystem's foundation.

1.3 Contribution

Virtual Service Platforms are a good solution but we still don't know how to really manage these components. We will develop an INDENICA Management Framework (IMF) in order to provide a working framework for the Virtual Service Platform proposed by INDENICA. The IMF will be designed as a domain-specific platform with emphasis on openness and interoperability and will address the problems introduced in Section 1.2. Being interoperable by design, the IMF can also be used to provide a virtualization layer for existing infrastructures.
One of the main IMF features is that its platforms can be modified after deployment. In this way it provides capabilities to make decisions, that are usually made at design time, at runtime. In general, the IMF will provide the following results:

- it vanishes the complexity caused by the fragmentation mentioned in the sections before. The reuse of functionality is one of the most important aspects that leads to a complexity reduction in platform ecosystems;

- it supports platform convergence and interoperability in order to vanish the dependency on external services and platform vendors by providing a common base for platforms that integrate system management and interoperability capabilities. The system management complexity will be reduced by making it reusable across the platforms. Capabilities in the system management will also comprehend components for dynamic adaptation of service in order to satisfy QoS requirements;

- it will be a reusable infrastructure provided by tools for supporting monitoring, governance and adaptation of services in a Virtual Service Platform. It will not be a service-infrastructure but a runtime architecture that can be integrated in a single interoperable platform. It will not be possible to develop a runtime platform in order to support the whole range of platforms for all the domains so we will develop the INDENICA Management Framework by focusing on a general scenario that covers all the way from embedded industrial control to high-end ERP systems.

In order to provide infrastructure components and tools to support the effective creation of domain-specific Virtual Service Platform, the IMF will be composed of two main elements: IMF Runtime and IMF Tools. The IMF Runtime is the real Virtual Service Platform that will provide capabilities of monitoring and adaptation. The IMF Tools is a GUI that will help the user to configure the whole IMF Runtime.

1.4 Organization

The reminder of this thesis is structured as follows:

- Chapter 2 details the current state of the art in the areas of SOA, publish-subscribe systems, autonomic computing, complex event processing, business rules management and model driven development. There will also be presented particular engines for complex event processing and business rules management.

- Chapter 3 provides an overview of related work in the area of systems' managing and adaptation in companies based on the service oriented architecture. In particular, we will describe some architectural designs and some developed frameworks.

- Chapter 4 we provide a general description of the INDENICA Management Framework. We will also describe all the design decisions. In particular we will introduce all the main INDENICA Management Framework functionality and components.

- Chapter 5 covers a presentation of the prototype implementation of the INDENICA Management Framework. In particular we will describe all the components developed and the technologies used in order to get the platform working and satisfy all the requirements.

- Chapter 6 contains an evaluation of the INDENICA Management Framework on a warehouse general scenario in which we had to cover embedded industrial control and high-end ERP systems requirements.

Chapter 2

State of the Art

The following chapter shows the state of the art of the technologies that helped the development of INDENICA Runtime Platform. In SECTION 2.1 *we talk about the base system architectural concept used as a model for the deployment of INDENICA Runtime Platform: SOA. Because of the complexity of Web Services, we need new ways to manage them so in* SECTION 2.2 *we talk about the autonomic computing. In* SECTION 2.3 *we talk about publish-subscribe system, one of the best solution to let these distributed systems communicate. The communication is based on events and in* SECTION 2.4 *we propose a way to manage those events. In* SECTION 2.5 *we talk about rule engines, a good way to transfer business rules in our system. In* SECTION 2.6 *we talk about how is possible developing application based on models.*

2.1 SOA

Most of the new programming languages use objects as paradigm but there wasn't a way to use these objects among different programming languages or platforms. In order to solve this problem, there were made new proposals like component-based technologies. These components encapsulated the objects and were able to serve different services, based on the objects properties, to the clients and this is the reason because they were called services. Service Oriented Architecture (SOA) is the architectural concept where business functions are used as services. When we design services, we have to consider some factors such as encapsulation (services should hide the implementation details), service contract (an agreement on how service and client will communicate and execute the operations), autonomy (the service should perform all the operations it offers in total autonomy), latency (be-

cause of the HTTP communication), etc. [17].
As shown in Figure 2.1, each component in the system can be discovered dynamically and can play one of the three following roles:

Service Provider : entity that creates a web service and possibly publishes its interface and access information to the service registry.

Service Requestor : entity that locates entries in the service broker using various find operations and then binds to the service provider in order to invoke one of its web services.

Service Broker : entity that acts as a repository for the interfaces published by the service providers.

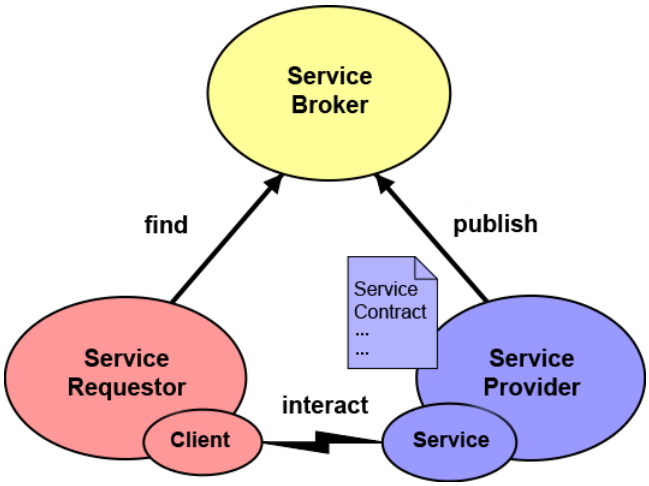

Figure 2.1: Service Oriented Architecture [29].

The general sequence is as follows: the service provider implements the service and publishes the service to the service broker; the service requestor discovers the service in the service broker and invokes the service in the service provider.
The main technologies used in the architecture are XML (data description), SOAP (service invocation), WSDL (service description) and UDDI (service discovery).
Simple Object Access Protocol (SOAP) is a lightweight protocol used to exchange information and independent from the platform, programming model, transport. There are three main parts of the SOAP protocol:

Envelope : describes what should be in the message and how it can be processed.

Encoding rules : rules to describe the encoding of the application data types.

RPC representation : representation of the remote procedure call. In particular, it is represented by a XML structure with the method name and all the method parameters are sub-elements in the XML message.

Web Service Description Language (WSDL) tries to solve a particular SOAP problem: it doesn't describe which kind of messages should be transmitted and where. So WSDL describes the communication in a XML structured way.
Universal Description, Discovery and Integration (UDDI) is a technology that enables to discover and find services. In this context, discovery means that you can discover the service and also get all the information to invoke it. UDDI is in general represented by a registry where it is possible to publish all the information about a new service or make queries about some services [13, 42, 56, 50].

2.1.1 Web Services

"*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format*" [13].
Some advantages that we can have by using the web services are:

- interoperability between different applications on different platforms;
- use of "open" standard protocols: data format and protocols are mostly text-based and this enables an easy development;
- use of HTTP as transport standard so we don't need to modify our firewall settings;
- can be used combined together (without any dependence from the provider and the location) in order to create complex and integrated services.

The main reason to implement a web service is the decoupling offered by the standard interface exposed by the Web Service itself. Any change on the applications running in the user system or in the Web Service doesn't affect the interface. This is a good property that enables to create complex systems made out of decoupled components and a strong usability of the

code.
A new architectural style to implement web services is REST (Representational State Transfer). REST is a hybrid style derived from several network-based architectural styles. The central REST feature is that it tries to have an uniform interface between the components and this distinguishes it from other network-based styles. There are four interface constraints that define REST: identification of resources, manipulation of resources through representations, self-descriptive messages and hypermedia as the engine of application state. In order to have the focus on the components' roles, the interaction constraints between the components and their interpretation of significant data elements, REST ignores the details of component implementation and protocol syntax. A *resource* is the key abstraction of information in REST. Thus, any information that can be named can be a resource (e.g., a document or image, a temporal service). REST components perform actions on a resource by using a representation to capture the current or intended state of that resource and transferring that representation between components. All REST interactions are stateless so each request contains all of the information necessary for a connector to understand the request, independent of any requests that may have preceded it [23].

2.2 Autonomic Computing

Actual company systems are characterised by the integration of heterogeneous environments into company-wide systems and extension of the boundaries into the Internet. This evolution led to more complexity in systems managing. In this scenario we are already over the human managing limits and architects cannot predict most of the interactions among the components because of their complexity. We need new ways to control these systems and the best way seems to be the one that lets themselves adapt by giving them high level rules. This is the aim of autonomic computing (a branch of artificial intelligence) even if the realisation of the solution won't be so easy and will involve many researchers in different fields. By self-management, autonomic systems can maintain and adjust their settings, demands, workloads and manage software or hardware failures.
In the autonomic system there are several components, each of them autonomic. As shown in Figure 2.2, every component consists of one or more managed elements and an autonomic manager. The autonomic manager is characterised by four components that represent the main activities in an autonomic system:

- Monitor: receive incoming data from the managed element and aggregates, collects, filters and reports details.
- Analyse: problem detection by modelling complex situations and understanding the current system state.
- Plan: adaptation choice in order to achieve goals and objectives.
- Execute: changes the behaviour of the managed element by executing actions on it.

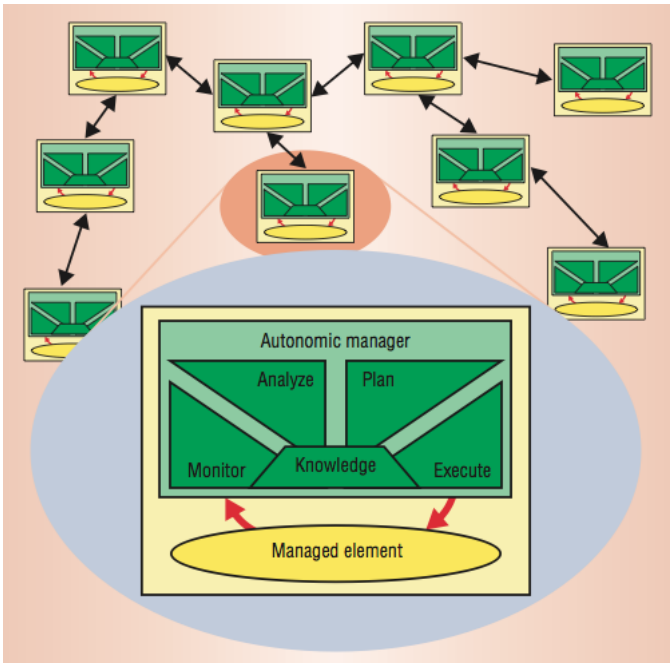

Figure 2.2: Autonomic MAPE System[34].

IBM found four main aspects of self-management [34]:

self-configuration : in the current computing we have data centers with different vendors and platforms so installing and configuring is time consuming and increases the probability of errors; in the autonomic systems all the configuration is automatic and follows high-level policies.

self-optimization : in the current computing there are a lot of manually set parameters and they could grow by the introduction of new releases; in the autonomic systems the components try every time to change these parameters in order to reach the best performance.

self-healing : understanding which could be the cause of a problem in the current computing could take weeks of programmers' work; in the autonomic systems the problems are automatically discovered and fixed.

self-protection : in current computing the detection and recovery from attacks or from cascading failures is manual; in the autonomic systems not only the detection and recovery are automatic but they also try to prevent system failures.

The acronym MAPE (monitor-analyse-plan-execute) aims to represent the four main activities executed by an autonomic manager: monitoring the managed elements and executing something on them based on the high-level rules set up. The managed elements represent the components without any autonomic rule and could be hardware or software. As shown in Figure 2.2, we can use multiple autonomic managers in a hierarchical fashion for multiple resources: low-level managers deal with resources at a smaller granularity and/or smaller locality, top-level managers can be used for business decision-making and/or policy and QoS levels. The automated rules planning process follows few steps like describing the domain, finding the initial state, creating some actions that can bring the system from the initial state to a state that satisfies the high-level objectives [48].

2.3 Publish-Subscribe

With the Internet the distributed systems are now made up of many entities around the world. The old way of communication between these entities, such as synchronous or point to point, is not working well anymore so we need a more flexible communication. Publish-subscribe seems that could fit the requirements for these new systems. As shown in Figure 2.3, in this model of asynchronous communication based on events there are three main actors:

Subscriber : expresses the interest on some events by subscribing on topics or defining some patterns.

Publisher : publishes events in the system.

Broker : event notification service that receives the events from the publisher and sends them to the correct subscriber(s).

There are different publish-subscribe variations because subscribers are often interested in particular events and not in all of them.

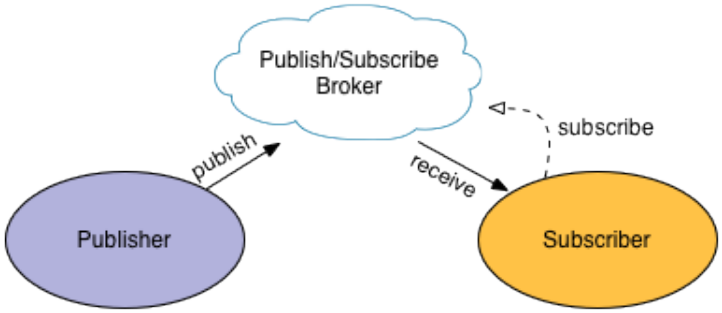

Figure 2.3: Publish-Subscribe System.

topic-based : the first implementation of publish-subscribe system where there are different spaces named by the topic keyword so publishers publish the event in the correct space and subscribers subscribe on one or more spaces. This works like the channels or we can see this method like one topic can be a group and all the components of that group can listen to that topic or say something on that group.

content-based : the events are no more classified based on an external criterion but on the actual attributes that they have. Subscribers can subscribe on contents that are defined by name-value pairs with comparison operators ($=$, $>$, $<$, $\geq$, $\leq$) where the name is the attribute name of the event.

type-based: events are filtered not by the topic but by their type. From one point of view it could be better because it enables closer integration with the language and we can ensure type safety at compile time. The reader can understand that type-based can easily become a content-based if we turn public all the attributes of the event.

There are several designs of publish-subscribe system and we cannot say which one could be the best one at all [21, 22].

2.4 Complex Event Processing (CEP)

Traditional computing uses databases that contain static data. In the realtime computing everything is based on streaming events. Complex Event Processing (CEP) is a set of techniques and tools used to help understanding and controlling event-driven information systems pioneered in late 1990's at Cambridge University in the UK, the California Technology Institute and Standford University.

Complex events are events that can only happen if lots of events happened. In the Event Driven Architecture (EDA) we have loose coupling because the creator of the event doesn't know who is going to *consume* it.
There are different event consuming styles:

Simple event processing : the event occurs and an action is initiated.

Stream event processing : stream of ordinary and notable events that are filtered to raise significant events.

Complex event processing : ordinary events of different types and longer time spans where the correlation can be casual, temporal or spatial.

In order to get it working we need a stream processing engine that could manage those events. We can run more that one of these engines in order to satisfy performance, scalability and fault tolerance requirements. The Complex Event Processing Language (EPL) is the language used in the CEP platform in order to define the logic to generate complex events. CEP engines give some benefits to businesses like identifying revenue opportunities in real time, quickly deploying and managing new services, detecting and stopping fraud as it happens, spotting favourable market conditions as they arise, reducing errors as they are made and detected [32, 44].
One of the first two CEP engines commercially available was Apama, based on the research by Dr. John Bates and Dr. Giles Nelson at Cambridge [36].

2.4.1 Esper

Esper [20] is a lightweight open-source CEP engine written entirely in Java and fully embeddable into any Java process-custom, JEE, ESB and BPM available under a GPL license. It is capable of triggering custom actions when event conditions occur among event streams. When millions of events are coming it would make it impossible storing them in a database and later query and that is why we need engines like Esper that can support high-volume event correlation.
Esper is able to handle events as JavaBeans, arbitrary java classes, java Map objects, or XML documents. The expression of conditions, correlation and spanning time windows is in Event Query Language (EQL): an object-oriented event stream query language very similar to SQL in its syntax but it significantly differs in being able to deal with sliding window of streams of data [4, 5]. An example of EQL query can be the following one:

```
select *
from StockTick(symbol='AAPL').win:length(2)
```

```
having avg(price) > 6.0
```

Listing 2.1: EQL Esper Query example.

In the Listing 2.1 we have the event class *StockTick* with attributes *String symbol* and *Double price*. This EQL query will trigger a new event every time average over the last two (*win:lenght()*) StockTicks having the symbol "AAPL" is above the value of 6.0.

2.5 Business Rules Management

A business rules[1] engine is a system that executes one or more business rules coming from legal regulation or company policies in a production environment. This system enables to define, test, execute and maintain the company policies separate from application code and typically supports rules, facts, priority, mutual exclusion and preconditions. In any IT applications, business rules change more frequently than the rest of the application code and in Figure 2.4 we can better understand which are the main components to manage them.

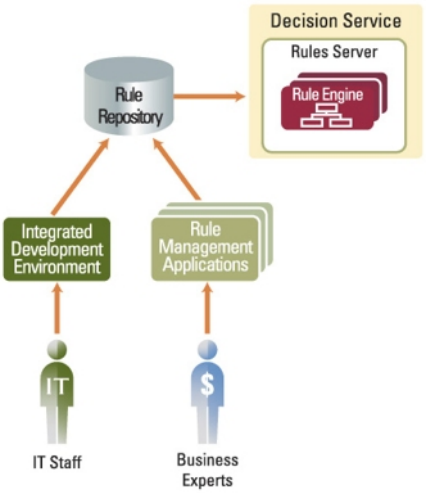

Figure 2.4: Common architectural components required to manage business rules.

Using rules can make easy expressing solutions to difficult problems and consequently have those solutions verified, they are much easier to read than code and we can create a single repository of knowledge (a knowledge base) which is executable. We have data and logic separation: data is in the domain objects and the logic is in the rules.

[1] A business rule is a statement that defines or constrains some aspect of the business.

Rule systems are capable of solving very, very hard problems, providing an explanation of how the solution was arrived at and why each "decision" along the way was made (not so easy with other of AI systems like neural networks or the human brain - "I have no idea why I scratched the side of the car"). The rule engines generally differ in how rules are executed:

production/inference rules : rules used to represent constructs like *IF condition THEN action.*

reaction/EventConditionAction rules : the reactive rule engines detect and react to incoming events and process events.

The main difference between these rules is that production rule engines execute when a user or application invokes them and a reactive rule engine reacts automatically when events occur. Most of the engines in the market support both rule execution ways [38, 53].

2.5.1 JBoss Drools

Drools[31] is a Java open source implementation of rule engine distributed under the Apache license [25]. The inference engine has the main role of matching facts and rules (pattern matching) to draw conclusions which ends in running actions. This matching is done by the RETE algorithm [24] extended by object oriented concepts.
Knowledge representation, which is a way of providing the rule engine with data, is based on first order logic (FOL)[2].
There are two main parts in a rule: condition(s) and action(s). As we can see in the Listing 2.2, the condition part is introduced by the *when* tag and the action by the *then* tag.

```
when
  Customer (age > 17)
then
  System.out.println("Customer is full age");
```

Listing 2.2: Drools rule example.

The condition in the Listing 2.2 is true for all those facts, which represents full age customers and could be similar to a SQL query in the Listing 2.3.

```
SELECT *
```

[2]First Order Logic allows to evaluate expressions like "2 + 3 == 5" or "customer.age > 17".

```
FROM Customers c
WHERE c.age > 17
```

Listing 2.3: SQL Query example similar to Drools rule in Listing 2.2.

The Drools rule engine gives some advantages that could be summarised in the following list:

- separates the application from conditions which control the flow: rules are stored in separate files, can be modified by different groups of people, changing rules does not require to recompile or to redeploy the whole application, putting all rules into one place makes it easier to control or manage the flow of the application;
- rules can replace complex if-then statements in an easy way: the rule language is not very difficult to learn, rules are easier to read and understand than code;
- problems are solved via rules, not by using a complicated algorithm: with declarative programming focuses on WHAT we are solving and not HOW, sometimes the algorithmic approach may be unusable (too complex, too time consuming, etc).

Drools is not only a rule engine but also an application for managing rules (Business Rules Management System). So we can create, modify, delete, branch and persist rules, assign roles to users (by using a login mechanism and LDAP integration for security) [35].

2.6 Model Driven Development

Developing complex software systems using code-centric technologies requires a lot of effort today. In particular, there is a big gap between the problem and the implementation domain.
Model Driven Development or Engineering (MDE) is one of the approaches that tries to decrease this gap. A model is an abstraction of some aspects of the system and that system may or may not exist when the model is created. MDE consists of creating abstract models of software systems and later concretely implement them. One of the main focuses of the MDE research is to produce technologies that separate software developers from the complexities of the underlying implementation platform.
We can distinguish two main classes of models:

Development models : software models at code abstraction level (requirements, architectural, implementation models).

Runtime models : abstractions of some parts of an executing system.

These models could also be used for more than just documentation during the software development. Technologies that raise the implementation abstraction level can improve productivity and quality while respecting the types of software targeted by the technologies.
The OMG [41] is an organization that develops and maintains standards for developing complex distributed software systems and launched the Model Driven Architecture (MDA) as a framework of MDE standards in 2001 [49]. As shown in Figure 2.5, the MDA separates business and application logic

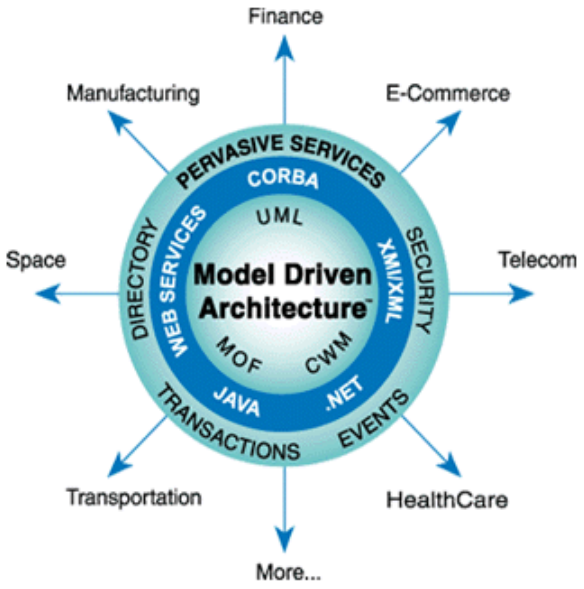

Figure 2.5: Model Driven Architecture (MDA)[41].

from the underlying platform technology but also tries to maintain separated the following viewpoints in the modeling systems:

Computation independent : focus on the environment in which the system will operate and on the required features of the system.

Platform independent : focus on the system features that don't change from one platform to another.

Platform specific : a view of a system in which platform specific properties are integrated with the elements of a platform independent model.

In conclusion, modelers need modeling methods that could provide them with guidelines to develop quality models. There are different ways to express

these guidelines like in the form of patterns, proven rules of thumb and exemplar models. The main problem is that the modelers usually ask for feedback from experts only at the end or when it is too late to determine the quality of their models [26].

Chapter 3

Related Work

During the years many researches were made in order to find and propose the best service oriented architecture for systems managing and adaptation in companies. In this chapter we will give an overview of all the main proposals. In particular, in SECTION 3.1 *we describe a service oriented middleware for context-aware applications. In* SECTION 3.2 *we describe another adaptive solution based on the SOA Solution Stack (S3). In* SECTION 3.3 *we describe a dynamic monitoring framework for the SOA execution environment. In* SECTION 3.4 *we introduce the COMPAS project. In* SECTION 3.5 *we introduce the QUA system. In* SECTION 3.6 *we introduce the SALMon platform. In* SECTION 3.7 *we introduce the MOSES framework. In* SECTION 3.8 *we introduce the VIDRE business rule engine.*

3.1 A Service Oriented Middleware for Context - Aware Applications

A relevant use of distributed systems is represented by context awareness. From a point of view the awareness makes the applications able to adapt their functionality when the contex changes, from another point of view the awareness raises new several requirements. A solution to manage all these new requirements could be enabling the applications to specify the major changes in the system.
Luiz Olavo Bonino da Silva Santos, Remco Poortinga - van Wijnen and Peter Vink proposed a context-aware middleware as integration of two components in a Service Oriented Architecture (SOA): the Context Management Service (CMS) [46] and the Awareness and Notification Service (ANS) [15]. The CMS has a publish-subscribe facility that enables the context sources to

publish information about their state in order to be used by context-aware applications and/or services. The ANS enables the client applications to define some context-based rules that are executed by a facility running in this component. It is also able to find the right context sources which it has to subscribe to in order to get all the context information for executing the context-aware rules.
As shown in Figure 3.1, the CMS has 3 main components: context source (provides information about a specific context and registers to the broker), context broker (saves and tracks all the context sources), context consumer (finds the right sources by asking the broker and uses their context information). From the same figure it is possible to understand the connection between the CMS and the ANS, as well as the ANS main components: event monitor (receives context information from the CMS and sends them to the controller), controller (receives events and evaluates the rules conditions), notifier (notifies the client application in case of rule firing), knowledge repository (stores all the rules), rule manager (enables the applications to define rules and stores them in the knowledge repository) [16].

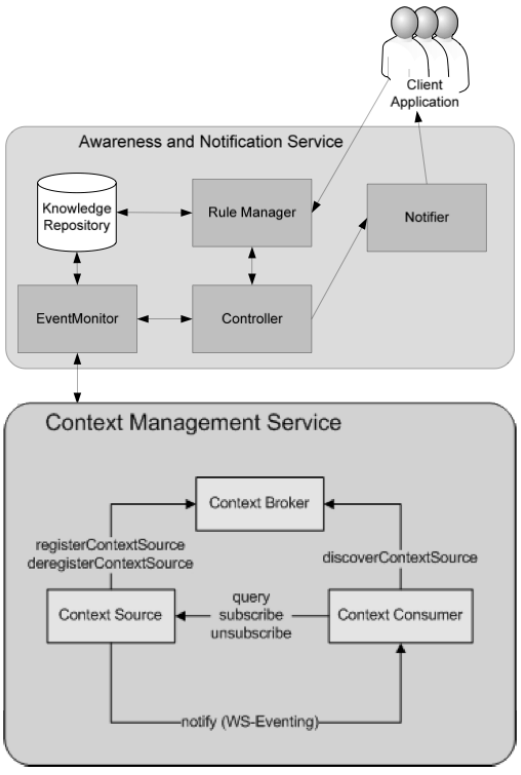

Figure 3.1: CMS and ANS main components and interaction.

3.2 Adaptive SOA Solution Stack

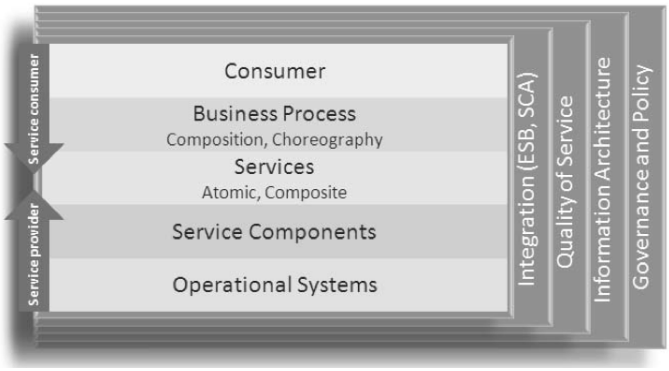

Figure 3.2: SOA Solution Stack (S3) model.

The aim of this work is to try to solve some issues still present in the SOA Solution Stack (S3) [2] context proposed by IBM, such as:

- Building a unique adaptive system model that defines interfaces, enables the building of an adaptation loop and could be used for any SOA system.
- Finding the best implementation technology that could better fit over the whole S3 layers. The implementation should consider also efficiency and scalability issues because the adaptation will process lots of data during the SOA application runtime.
- Because of the SOA systems dynamicity, we need this property also in the adaptation by providing reconfiguration on demand (thus, without any suspension or restarting of the system).
- The adaptation strategies fired at the same time could be more than one so we need to adopt statistical/probability theories in order to execute the best one.

The researchers of the AGH University of Science and Technology integrated adaptation features to the S3 model shown in Figure 3.2 by developing the Adaptive S3 (AS3) element. In Figure 3.3 is shown an AS3 element with all its components working in a single S3 layer. The resource represents anything that could be monitored and managed, and send information data by sensors. The sent data is collected by the monitoring component and it decides to create single events for each data or aggregate them in a complex one. In this component can run CEP engines (refer to Section 2.4) like ESPER

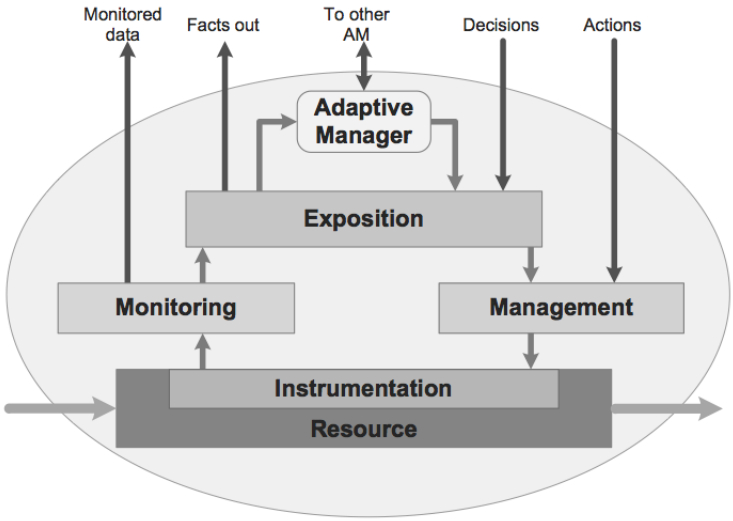

Figure 3.3: AS3 element model.

or DROOLS. The exposition component receives these events. It has an adaptive manager that selects the action to be performed and, in case of firing of some rules, it will send the information about the adaptation strategy to the management component. The management component will thus execute the adaptation actions on the resource. As we said, the Figure 3.3 shows the application of AS3 in a single S3 layer but it is easy to understand that the information about the resource can be sent in different layers of S3 architecture where different AS3 elements are running [57].

3.3 Dynamic monitoring framework for the SOA execution environment

Service Oriented Architectures (SOA) are increasingly used in order to face the constant changes in business requirements at runtime. But we need to define some requirement for SOA management if we want a service composition to be dynamically rearranged without shutting down the services.
Daniel Żmuda, Marek Psiuka and Krzysztof Zielinski propose a framework that can satisfy these needs. The frameworks uses two main elements of the Event-driven SOA [52]: event-based system and complex event processing engine. The adapted interceptor pattern is the design chosen for this framework: interceptors are exposed as services and they can provide on-demand monitoring data to the monitoring environment. The interceptors that need to be plug are determined by the monitoring scenario that could also be

dynamically changed. The middleware platform used as Enterprise Service Bus (ESB) is an OSGi ESB container [58].

3.4 COMPAS

Researchers involved in the COMPAS (Compliance-driven Models, Languages, and Architectures for Services) [7] project, developed a service oriented framework that could manage company compliance. In order to get the framework running, the COMPAS project also provides a tool define the compliance governance lifecycle and its output is a BPEL process of the process model. The COMPAS runtime framework (Figure 3.4) takes as input this BPEL process and deploys it to the Extended Process Engine Apache ODE. The process UUDI is published to the Process Engine Output (a Java Message Service Topic) in the Enterprise Service Bus Apache ActiveMQ. From the other side, there is an Advanced Telecom Service Custom Controller (ATSCC) subscribed to this topic with the role of selecting pre-defined events. All the events that pass the check are then published in an other JMS-Topic (Compliance Govern Input) which are subscribed to the CEP and the Event Log engines. The CEP has the role of finding complex event patterns from the incoming events. In this way system-level events are transformed in business-level events. The results are shown online on the Compliance Governance Dashboard while the business-level events are published to the JMS-Topic CEP Engine Output. These events are stored into the Event Log component, as the system-level events. The Event Log is accessed by an ETL that extract, transforms and loads the data into the Data Warehouse that is then extracted and analysed by the Analysis/Business Intelligence component. Also the results of this offline process are finally shown in the Compliance Governance Dashboard.

3.5 QUA

The main approach used to build systems from compositions of inter-organisational services is Service Oriented Architecture (SOA). The fragmentation of these services is still present because of the different development technologies, published by different providers, etc. Also the application logic could be divided in two layers (T. Erl [19]): the service interface layer (services communicate via open protocols hiding the implementation) and the application layer (application logic developed on different technology platforms). SOA-based system should be able to adapt at the layer(s) in order to adapt

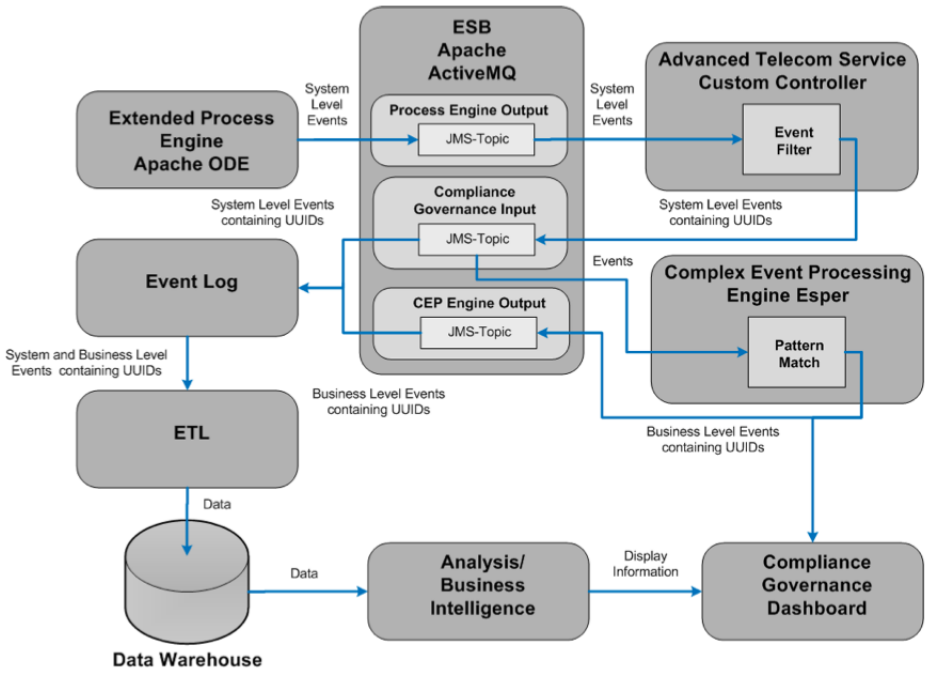

Figure 3.4: Runtime compliance governance architecture.

to context changes at runtime so we need an adaptation framework in both layers, possibly without leading too much re-implementation or re-factoring. An adaptation framework that could be used on SOA-based systems to perform cross-layer adaptation is QUA [27]. Even if QUA is used across different layers, it could also be applied on the two layers separately. The framework is composed by a planning framework (responsible for choosing service implementations and configurations) and a platform framework (able to manage and adapt services).
In Figure 3.5 there is a design view of the cross-layer middleware build by using the QUA. It is easy to understand that the adaptation mechanisms used in the two layers are different because of the layers nature. In order to integrate the two layers, the first step to do is mapping the concepts and artefacts of the two layers into a common QUA meta-model. The planning framework will then use this information to understand the dependencies between the service interface layer and the application layer. The second step is developing a technology specific platform that encapsulates runtime environments and adaptation mechanisms. All the information needed to instantiate services and adapt them are received by each platforms from the planning framework [28].

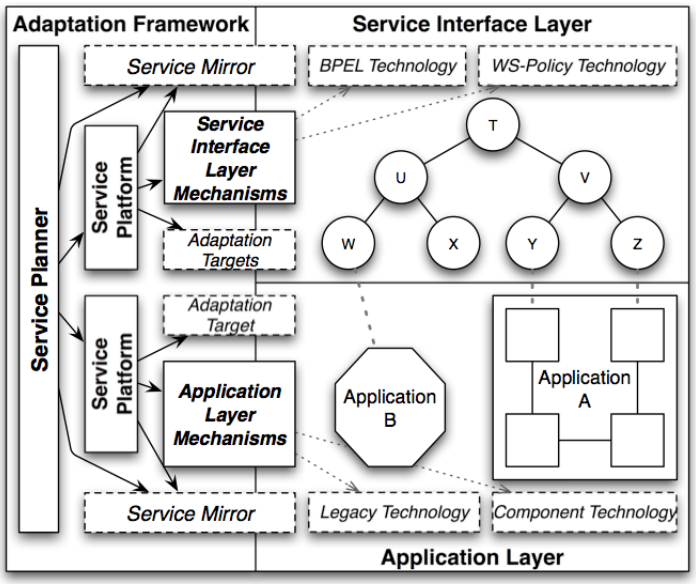

Figure 3.5: Design of the cross-layer adaptation middleware.

3.6 SALMon

In order to fulfil the QoS requirements and the SLAs, self-adaptive systems can enable services to change dynamically. SALMon is a SOA system that provides QoS information at runtime useful to detect SLA violations. The platform is composed by the three main services shown in Figure 3.6. The Monitor service uses the Measure instruments in order to obtain information about Qos. The Measure instruments communicate with the services and get monitoring information from them. This information are then rendered by the Monitoring service and transferred to the Analyzer service. This component checks SLA violations and when a violation happens, it notifies the Decision Maker service running in the service affected by the violation. The Decision Maker service chooses the best operation to do in order to solve the problem in the service. This solution lacks scalability because each Decision Maker service runs in one and only one service [43].

3.7 MOSES

In order to react to changes in the operating environment, we need some tools to enable SOA systems to self-adapt. MOSES (MOdel-based SElf-adaptation of SOA systems) is a tool developed by people from the Italian University of Rome and Technical University of Milan that can allow self-

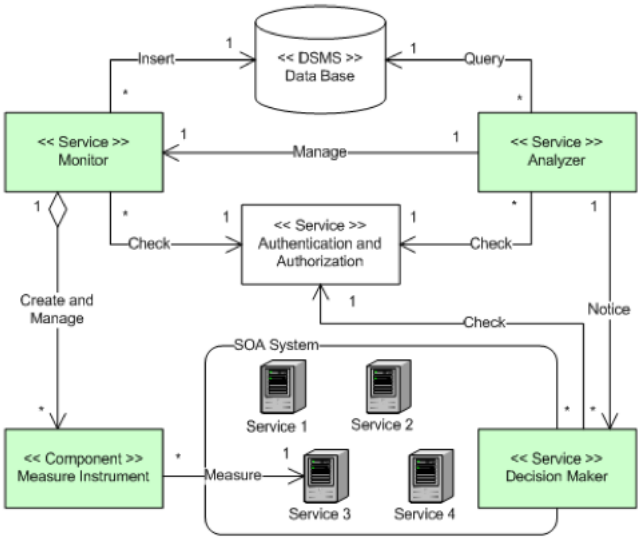

Figure 3.6: SALMon Platform Architecture.

adaptation in these SOA systems.
Sometimes it could happen that there is not any combination of services that can fulfil the QoS requirements but MOSES proposes multiple independent implementations of the same functionality in order to solve it. Thus, instead of considering concrete services, it binds abstract services to a set of concrete services. MOSES enables multiple user definition of QoS requirements and it will try to have a runtime platform that satisfies an average of the duplicated requirements.
In Figure 3.7 is possible to see the core MOSES framework's elements and their connection. MOSES takes as input the description of the composite service in the BPEL [33] workflow orchestration and a set of services candidate to fulfil the requirements. In case that MOSES finds a model that satisfies the workflow, it builds a behavioral model of the composite service and passes it as input for the adaptation modules. The adaptation derives all the parameters from the defined SLAs. The monitoring activity has the role to give all the realtime information needed by the adaptation component in order to satisfy the SLAs even if the context changes at runtime. When an adaptation is needed, MOSES builds a new workflow with new parameters that satisfy the SLAs [9].

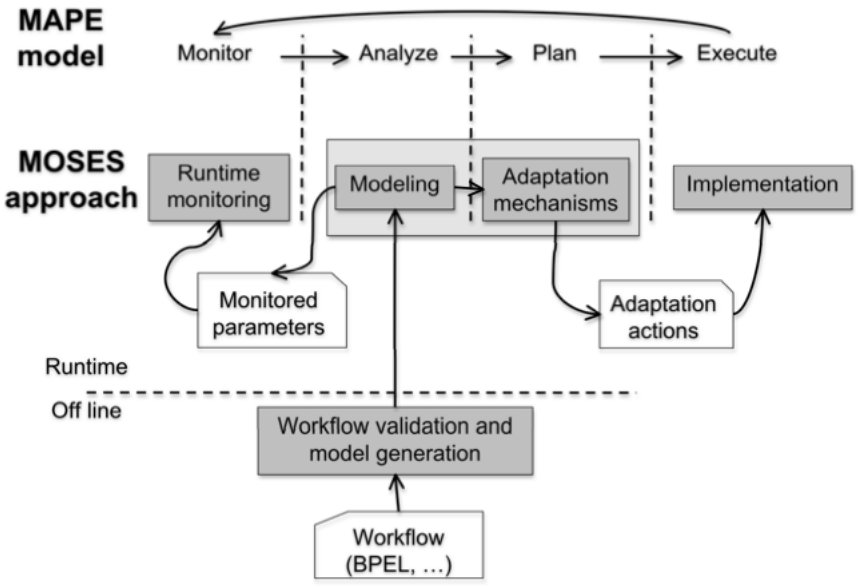

Figure 3.7: MOSE Platform Architecture.

3.8 VIDRE

VIDRE [40] is a business rule engine with several new features that can be summed in the listing below.

- It combines the rule-driven approach with SOA. This features enables both integration in client applications and the using of Web services that execute business rules.
- It makes easy the communication between the client and the business rules by using a plug-in mechanism. This feature makes also transparent the business rule engines exchanging to the client applications. The rule markup used is RuleML [47] that is supposed to be an easy language to define business rules as long as it is a standardisation of the rule markup.
- It can execute distributed business rules (execution of business rules over several rule engines).

Every business rule engine in VIDRE is contained by a VIDRE service provider (VSP) and its architecture is based on the Java Rule Engine API (JSR 94) combined with RuleML. In Figure 3.8, every VSP offers a generic RuleML interface that takes as input only valid RuleML documents sent by HTTP, JMS or SMTP. Every VSP publishes a WSDL interface in order to

access the client runtime and another WSDL interface to access the administration. There are two ways that the clients can use to access the VSP: by its Web service interface or by a SOAP-RuleML gateway that offers a way to access the business rules as Web services.

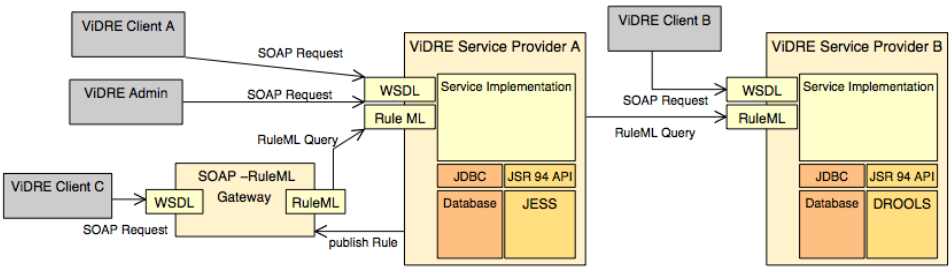

Figure 3.8: VIDRE Platform Architecture.

Chapter 4

Design

The following chapter introduces the INDENICA Management Framework. In particular, Section 4.1 *gives a short description of this thesis work. In* Section 4.2 *we show the architectural design of INDENICA Management Framework.*

4.1 Description

INDENICA Management Framework is proposed as a virtual platform tool for large systems in different areas. This tool will be useful in managing variability on a system wide level by covering the aspects of vertical and horizontal integration of different platforms.
The approach used by the IMF is goal-oriented: it addresses the challenges by describing requirements of the whole system. It will also provide views to describe the complete system architecture that can be used to generate connectors between the different platforms covered by the IMF. The Figure 1.1 represents a Virtual Service Platform (VSP): a special kind of service platform useful to hold the heterogeneity of the underlying service platforms and provide the right abstract layers to the service-based application developers. The major features in the final framework will be monitoring and adaptation functionality for the single platforms and the system as a whole.
Usually the functional and non-functional properties of a service platform and its interface vary with the requirements of a domain so we propose the IMF as a domain-specific service platform. In this way, customers can benefit from reusable software elements because software development is complex, time consuming and expensive.

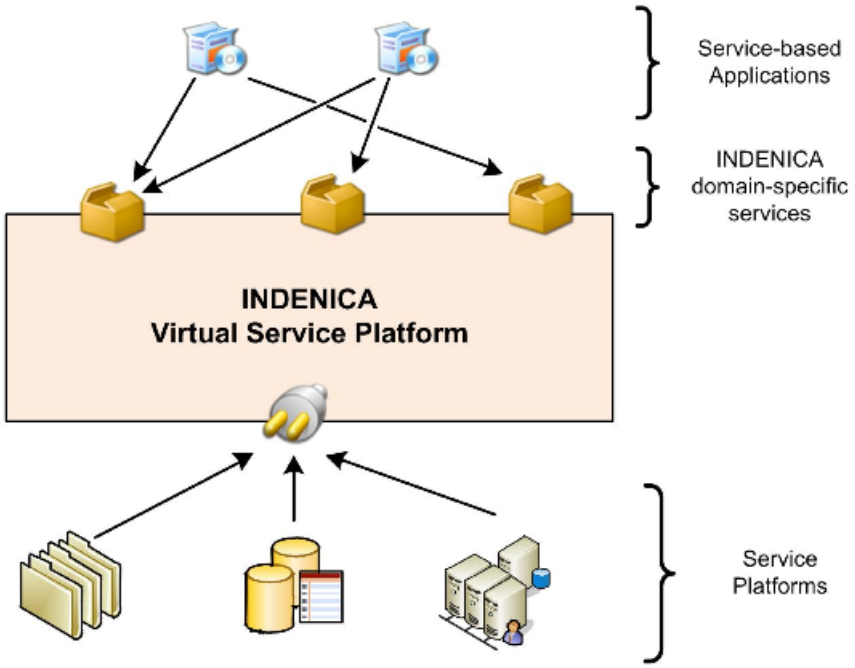

Figure 4.1: Overview of an INDENICA Virtual Service Platform.

4.2 Main System Components

Modularisation has been known as a good method in system engineering for many years in several scenarios going from IT systems to embedded devices. The IMF needs to support and enable the communication between these modules located in different layers of the Totally Integrated Automation (TIA) Pyramid shown in the Figure 4.2, in a range that goes from the IT systems used in the Enterprise Resource Planning (ERP) down to the controllers used in the Programmable Logic Controller (PLC) layer. Thus, we will adopt a modularisation technology for the IMF in order to satisfy the requirements and we will use the Service-Oriented Computing (SOC) [45] as a development pattern in the IMF in order to achieve software reusability. This means that services are used as the fundamental elements for the application development. The Service Component Architecture (SCA) is a standard of the Open SOA Consortium [12] and integrates the service-oriented paradigm with component-based development (for more details we suggest reading [10]). As shown in Figure 4.3, SCA components are the building units of modules and composite applications that communicate with each other via services. As far as SCA is standardized, supported by several vendors

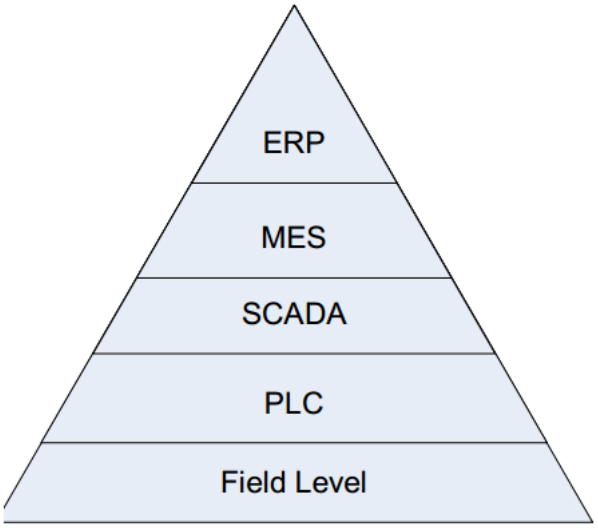

Figure 4.2: Totally Integrated Automation Pyramid.

and enables a high level of integration, it represents a potential technology for the IMF.

We can obtain several benefits by using SCA such as "rapid development

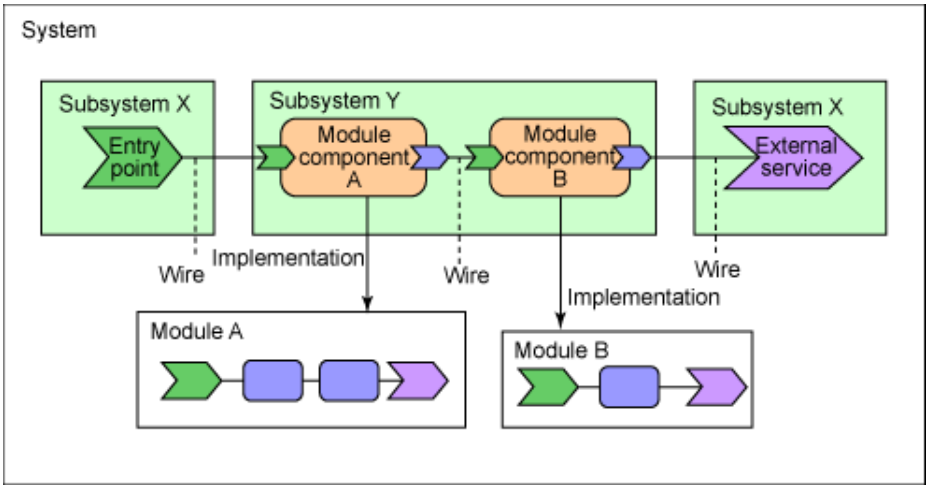

Figure 4.3: Composing components to modules in SCA.

and increase in productivity, higher organisational agility and flexibility, return on Investment through reuse" [37]. This architecture is also supported by many reliable open source platforms such as Apache Tuscany [1] or Fabric3 [51]. The IMF will accommodate domain-specific variability in non-functional requirements of platform and application components and will

also automatically compile domain-specific non-functional requirements into runtime policies (e.g. to be used for dynamic instantiation or migration of components).
A complete overview of the IMF is shown in Figure 4.4. In this way, the monitoring and adaptation framework introduced will significantly improve the lifecycle management of Virtual Service Platforms.

There will be developed all the components placed in the IMF Runtime

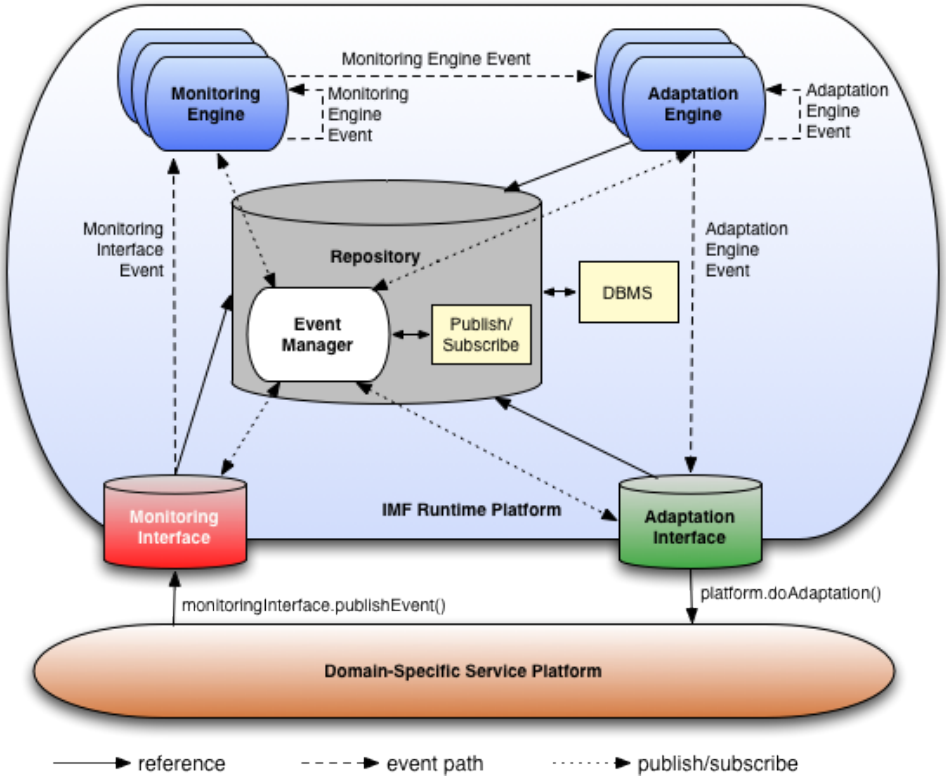

Figure 4.4: Overall architecture of INDENICA Management Framework framework.

platform while supporting models and model instances are marked in yellow color. In the following subsections we will describe better all the functionality for each component.

4.2.1 Repository

The central element of the figure is the Repository component. It will be developed as a service that could offer all the functionality of static storing to all the clients connected to the IMF and all the other components having a role in the platform. It will also store models which define how the platform variants differ from each other. Moreover, the Repository offers functionality

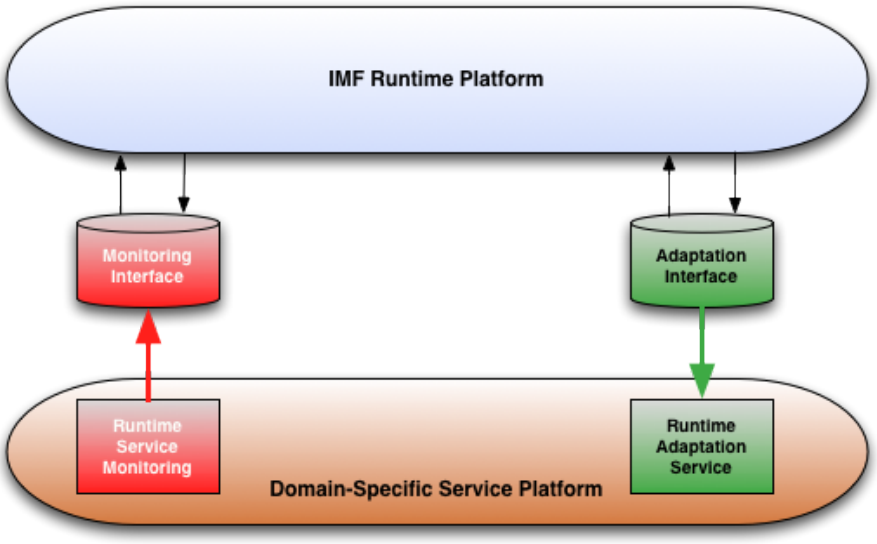

Figure 4.5: INDENICA Management Framework integration interfaces.

of managing runtime events generated in the IMF by platform's components or by the clients. Thus, the Repository acts as both the back-end database (by using a DBMS) and the online caching storage (by using a publish-subscribe framework).

4.2.2 Monitoring Engine

This component offers functionality of monitoring the connected system by using a CEP engine (Chapter 2.4) that catches proper events or sequences of events. These monitored events can come either from the monitored system or from the running Virtual Service Platform (VSP) instance. Thus, the purpose of Monitoring Engine is to fulfil the functional and non-functional requirements by providing feedback to the Adaptation Engine.

4.2.3 Adaptation Engine

This component receives events from the Monitoring Engine and offers functionality of adaptation to the monitored system based on previously defined rules and policies in a rule engine (Chapter 2.5). In this way, we will fulfil all the business rules defined.

4.2.4 Interfaces

In order to connect the monitored system to the IMF, we offer several services that can receive monitored system information and place adaptation actions on the monitored system. Figure 4.5 shows how the monitored system is integrated in the IMF by using these interfaces. In particular, the monitored system will provide services for a monitoring service to send monitoring information and an adaptation service to receive adaptation actions.

4.2.5 Configuration GUI

This component should provide the system configurator all the tools to configure the IMF for the best connection to the monitored system and the best governance of it. So there will be interfaces for the environment configuration, events definition (incoming or outgoing), CEP rules and business rules.

Chapter 5

Implementation

The following chapter shows the real implementation of the INDENICA Management Framework components. In Section 5.1 *we show the Repository implementation and the Event Manager (a particular component for events management) implementation. In* Section 5.2 *we show the Monitoring Engine implementation. In* Section 5.3 *we show the Adaptation Engine implementation. In* Section 5.4 *we show the Interfaces implementation. In* Section 5.5 *we show the Configuration GUI implementation.*

5.1 Repository

The Repository is a SCA component implemented in Apache Tuscany [1] for Java. All the components interested in the retrieval of some static information such as configuration parameters or storing of some data can connect to this component by a Tuscany *reference* or by invoking the service. This component has a connection to the real database implemented using MongoDB[39]. All the IMF Runtime configuration is in the *AdminDB* database in MongoDB and here are the collections (tables):

envConf all the IMF environment parameters.

adaptationPolicies all the adaptation policies in Drools language for the Adaptation Engine.

eventsIn all the definitions for the events incoming to the IMF Runtime.

eventsOut all the definitions for the events outgoing from the IMF Runtime.

monitoringRules all the monitoring rules in Esper language for the Monitoring Engine.

5.1.1 MongoDB

MongoDB is a document-oriented database so that documents (objects) map well to programming language data types and embedded documents reduce need for joins. The low need of joins leads to high performance: fast reads and writes. It also offers easy scalability capabilities like automatic sharding (auto-partitioning of data across servers) so reads and writes are distributed over shards. MongoDB data model is based on a set of databases; each database has a set of collections; each collection has a set of documents; each document has a set of fields; each field is a key-value pair. It offers also a rich query language represented by JSON-style queries [11]. An example of these queries by using the command line *mongod* server program is as follows:

```
db.things.find( { x : 3, y : "foo" } );
```

This is like the SQL query

```
select * from things where x=3 and y="foo"
```

5.1.2 Events Manager

In order to give all the runtime capabilities to the Repository, we decided to implement the Events Manager component: a publish-subscribe system that could manage all the events used for the platform components communication. All the components that want to publish or subscribe for an event need to know all the information about this system. This information can be obtained by connecting to the Repository and querying the configuration table. The publish-subscribe system is implemented in a server running RabbitMQ[55] message broker by using its topic based delivery capability. The main advantage of using a publish-subscribe system reached in our platform is that we have more flexibility on the monitoring and adaptation engines instantiation and everything is transparent to the components that publish some events.

RabbitMQ

RabbitMQ is a message broker so, the main idea is that it accepts and forwards messages. We can compare this system to a post office and the RabbitMQ broker is the metaphor of a post box, a post office and a postman.

A program that sends messages is a *producer*. A *queue* is the name of the mailbox and lives in RabbitMQ. A *consumer* is a program that waits for messages to be received.
The core idea in the messaging model in RabbitMQ is that the producer never sends any messages directly to a queue. Quite often the producer doesn't even know if a message will be delivered to any queue at all but can only send messages to an *exchange*. The exchange is like a proxy: receives messages from producers and pushes them to queue(s). There are different exchange types but we will focus on the *topic* one that is what we used in our publish-subscribe system. Messages sent to a topic exchange have a list of words, delimited by dots (e.g. "events.mi.#", "*.ae.usageHigh", "quick.orange.rabbit"). There are some special characters in the words:

***** can substitute for exactly one word

\# can substitute for zero or more words

A full example of Java code in the publisher side is in the Listing 5.1.

```
1 channel.exchangeDeclare("exchange_name", "topic");
2 channel.basicPublish("exchange_name", "myTopic.hello",
     null, "Hello");
```

Listing 5.1: RabbitMQ publisher Java code example.

The message will be received by the consumer that runs the Java code in the Listing 5.2.

```
1 queueName = channel.queueDeclare().getQueue();
2 channel.queueBind(queueName, "exchange_name", "
     myTopic.hello");
3 QueueingConsumer consumer = new QueueingConsumer(
     channel);
4 channel.basicConsume(queueName, true, consumer);
5 QueueingConsumer.Delivery delivery = consumer.
     nextDelivery();
6 String message = new String(delivery.getBody());
```

Listing 5.2: RabbitMQ consumer Java code example.

5.2 Monitoring Engine

In order to implement the monitoring aspect of the IMF Runtime we developed a Java component named *Monitoring Engine* 5.1. It processes the

received events from the monitored platform by using the Esper CEP engine. The events are received from the Event Manager which the Monitoring Engine subscribes to. The Esper processed events are then sent again to the Event Manager. All the information about which events it should subscribe to or send and the monitoring rules are retrieved by connecting to the Repository component. As shown in Figure 5.3, it is possible to instantiate more than one of these engines in order to have different hierarchy levels of monitoring. The Monitoring Engine will be also responsible for self-monitoring of the IMF Runtime by handling the right events from the Adaptation Engine. This is possible by subscribing on particular event patterns in the Event Manager (e.g. *events.monitoring.#*).
The basic assumption is that the Monitoring Engine should be able to receive and understand events from any current or future monitored platform. In order to offer this capability, we developed a generic and extensible Monitoring Event Model (*Event* used to convert events generated by the monitored platforms to internal IMF events. The Event model class can be found in the attached source code.

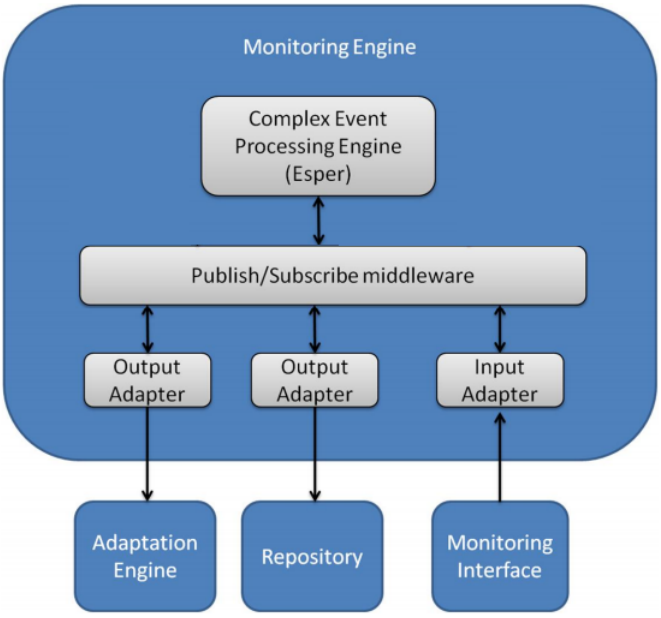

Figure 5.1: Monitoring Engine overview.

5.3 Adaptation Engine

The adaptation aspect of the IMF Runtime is implemented in a Java component named *Adaptation Engine* 5.2. It processes the received events from the Monitoring Engine (generated by Esper) by using the JBoss Drools engine. The events are received from the Event Manager which the Adaptation Engine subscribes to. The adaptation actions received from the Drools engine are transformed in the IMF events and sent again to the Event Manager. All the information about which events it should subscribe to or send and the monitoring rules are retrieved by connecting to the Repository component. It is possible to instantiate more than one of these engines in order to have different hierarchy levels of adaptation and each of them implements autonomic MAPE managers. This facilitates separation of concerns allows the low-level Adaptation Engines to deal with granular changes in system behaviour and the high-level Adaptation Engines to focus on the specification of overall service level goals.

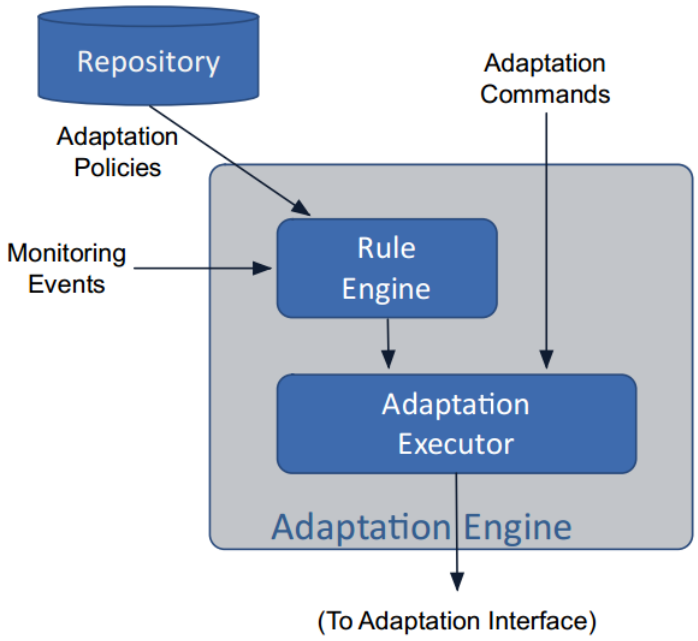

Figure 5.2: Adaptation Engine overview.

5.4 Interfaces

In order to offer this monitoring and adaptation service to the governed services we developed two different interfaces which they have to communicate

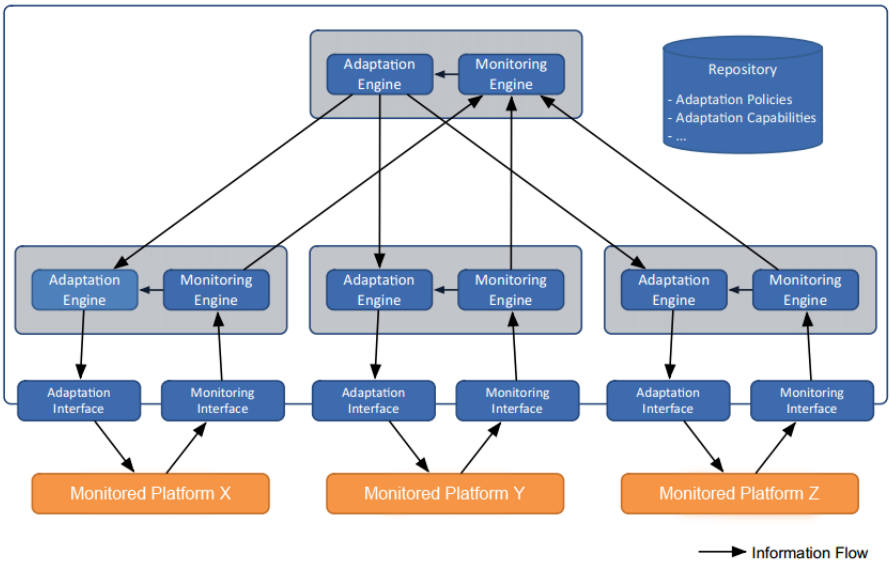

Figure 5.3: Hierarchy style of Monitoring and Adaptation Engines.

with: *Monitoring Interface* and *Adaptation Interface*.

5.4.1 Monitoring Interface

The Monitoring Interface is a SCA component implemented in Apache Tuscany for Java. This component is an integration interface for monitoring which enables various platform providers to communicate with the IMF Runtime. In particular, it offers to the monitored platform an interface for publishing events to be processed by the IMF Runtime Monitoring Engine. As shown in Figure 5.4, this component is divided in a client element (has to be integrated with the monitored service platform) and a server element (receives the monitoring events from the governed platform, it's the real Monitoring Interface). The monitoring client receives events from the monitored service platform and translates them into the IMF event model. Each platform provider will be responsible for defining a mapping between the events generated by their platform and the events model of the IMF. This mapping can be defined by using the Configuration GUI tools. After that the monitoring server receives these events from the monitoring client, it sends them to the Monitoring Engine by using the Event Manager.

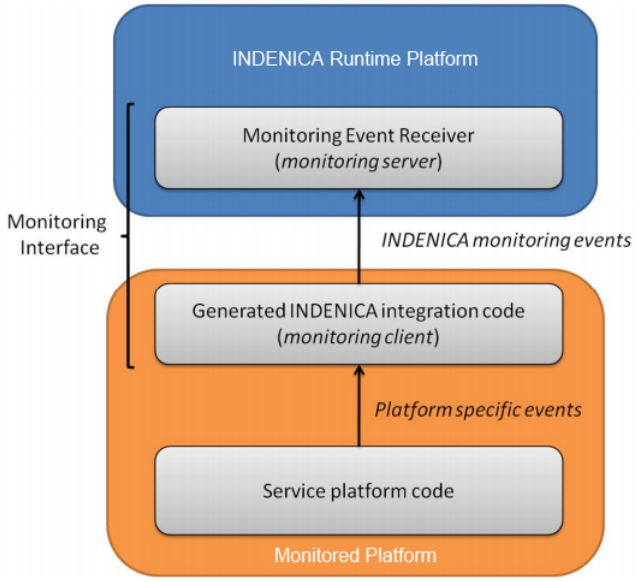

Figure 5.4: Monitoring Interface.

5.4.2 Adaptation Interface

The Adaptation Interface is a SCA component implemented in Apache Tuscany for Java. It calls the right monitored platform interfaces in order to place the adaptation actions fired by the Adaptation Engine. Platform providers specify in the Adaptation Interface the adaptation capabilities of their monitored platforms and map the IMF Runtime adaptation commands to platform-specific actions. As shown in Figure 5.5, the Adaptation Interface is divided in a component that is a part of the IMF Runtime and an adaptation commands translator that is located on the target monitored platform.

5.5 IMF Tools

In order to offer a guided and structured configuration framework of the IMF platform to the system configurator, we provide a GUI that can support all these operations: IMF Tools. In the Appendix A we provide also a user manual for the IMF Tools. These framework enables the user to configure all the base parameters and the communication between the components in the IMF Runtime in a easy way and without any big effort.

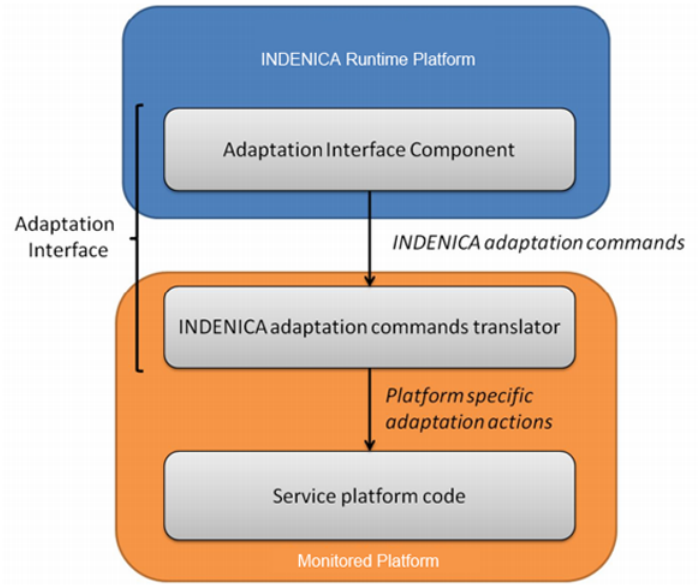

Figure 5.5: Adaptation Interface.

5.5.1 Main Selector

The Main Selector is the main GUI that enables the user to launch all the other tools. Moreover, it enables the user to connect to the Repository and save all the new configuration parameters set up by using all the other GUI tools. It has also a feature to populate the Repository with the initial parameters and a menu that drives the user to launch the right configuration tool.

5.5.2 Environment Configurator

The Environment Configurator is a tool that helps the user to insert all the parameters needed for the right working of all the IMF Runtime platform like defining the service platform name, the engines, setting up the tables where the components can find the right configurations, rules and the Event Manager connection parameters.

5.5.3 Incoming Events

The Incoming Events is a tool that helps the user to define all the events coming in the IMF Runtime platform from the monitored service platform. In particular the user can define the event type and a set of parameters in the form key-value where the key is the attribute name and the value is the

attribute type.

5.5.4 Monitoring Engines

The Monitoring Engines is a tool that helps the user to define the Esper rules that should be executed in each Monitoring Engine. The information about the Monitoring Engines IDs are retrieved from the environment configuration collection in the Repository.

5.5.5 Outgoing Events

The Outgoing Events is a tool that helps the user to define all the events coming out from the Adaptation Engine to the Adaptation Interface. In particular the user can define the event type and a set of parameters in the form key-value where the key is the attribute name and the value is the attribute type.

5.5.6 Adaptation Engines

The Adaptation Engines is a tool that helps the user to define the Drools rules that should be executed in each Adaptation Engine. The information about the Adaptation Engines IDs are retrieved from the environment configuration collection in the Repository.

5.6 UML Class Diagrams

In this section we show the UML class diagrams for the main classes of the IMF. We talk about *Runtime* when we refer to the main IMF Runtime platform components so we separate them from the IMF Tools.

5.6.1 IMF Runtime

The UML class diagram in Figure 5.6 contains all the main core components of IMF Runtime platform. All the interfaces of these components implement the *WP4Component* interface. This is because we want to have some common features in all the components. All the classes colored in yellow are the real implementation of the respective interfaces colored in white. We wanted to do this for two main reasons: the first one is because of the OO programming principles, the second one is because in Apache Tuscany all the *references* are interfaces and at runtime it starts the actual implementation class.

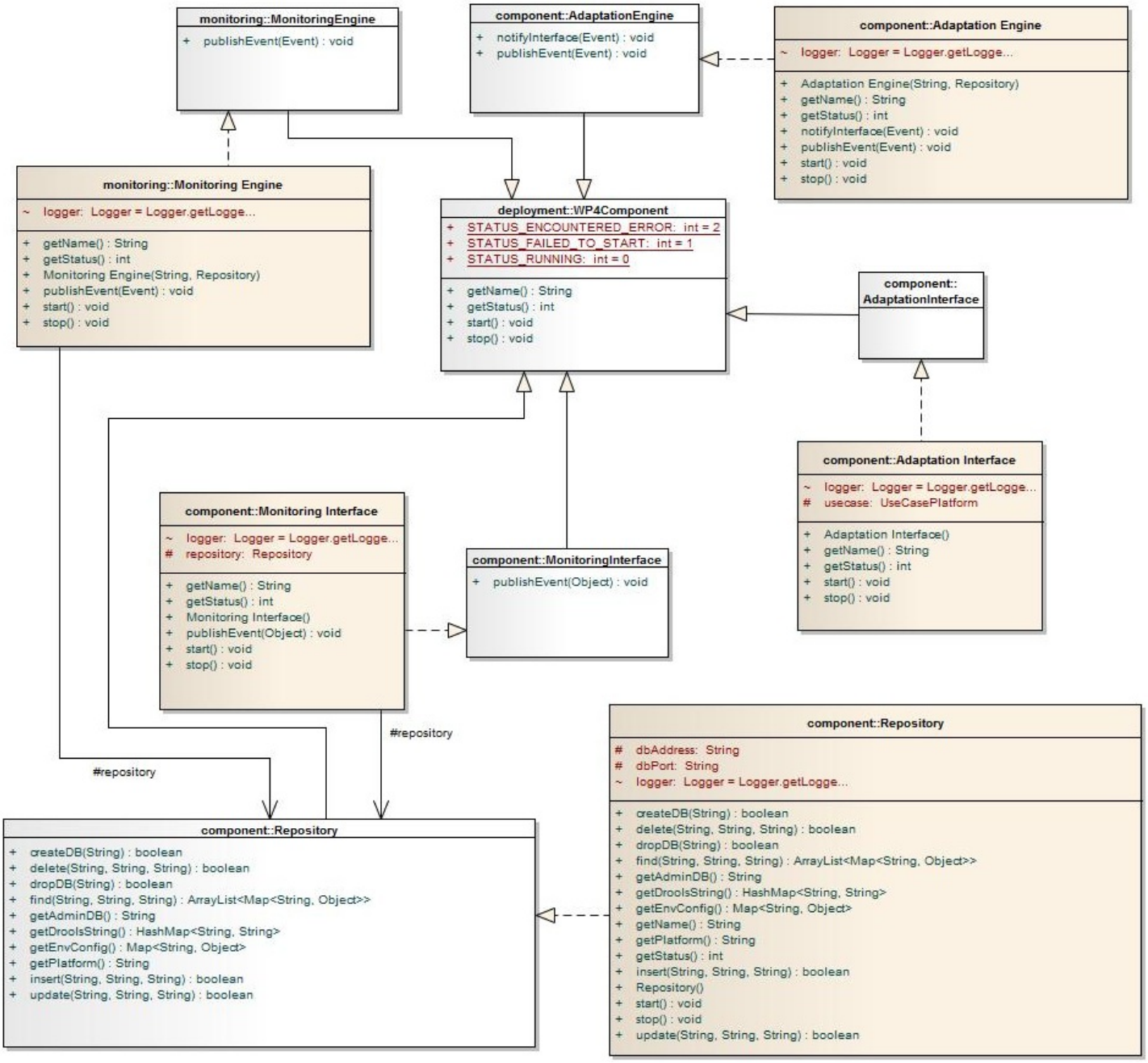

Figure 5.6: UML class diagram IMF Runtime.

5.6.2 IMF Tools

The UML class diagram in Figure 5.7 contains all the main classes used to develop the IMF Tools. This framework is a GUI that offers different screens able to help the user to define the IMF Runtime platform properties, the incoming events, the monitoring rules and engines, the adaptation policies and engines, and the outgoing events. For explanations about the usage, please refer to Appendix A.

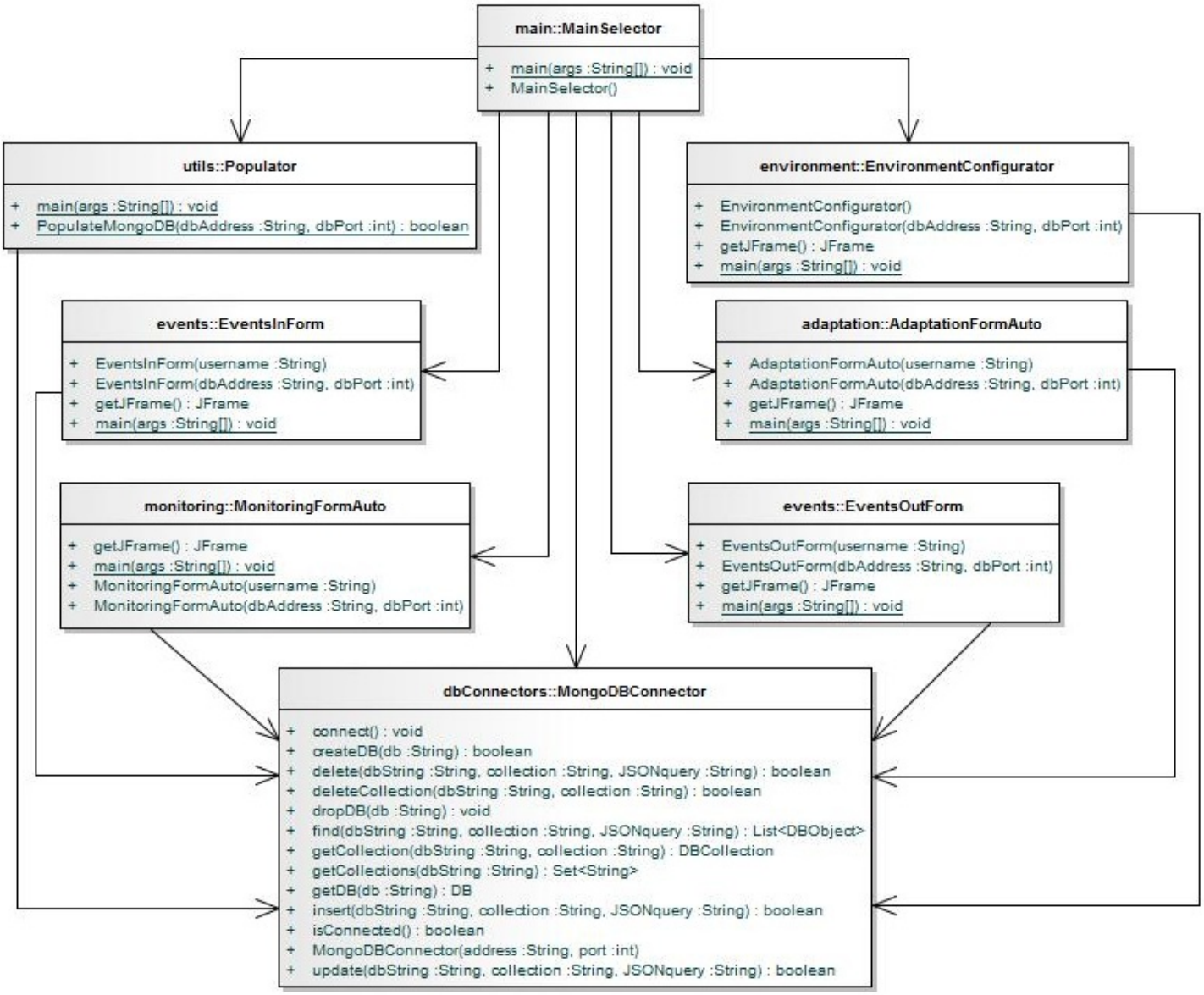

Figure 5.7: UML class diagram IMF Tools.

Chapter 6

Evaluation

The following chapter shows a feasible case study in order to evaluate the INDENICA Management Framework performance. In Section 6.1 *we introduce the warehouse case study and its main components. In* Section 6.2 *we show the evaluations results obtained by running the INDENICA Management Framework in the warehouse case study.*

6.1 Warehouse case study

Realistic requirements are the key to a successful development of research prototypes. The IMF tries to be a virtual platform for large systems spanning different application areas as well as different levels of the automation pyramid. A Warehouse Management System (WMS) is a well-known example from the industry automation area and this is the main scenario used as a case study to demonstrate the feasibility of the approach.
It is possible to recognise the main external parts of a warehouse by looking at the Figure 6.1. In particular, there are trucks in the yard waiting for a loading/unloading job, the staplers carrying the goods, the racks with storage units (bins) that are picked and placed by automated stapler cranes. It is also possible to recognise conveyers used to carry the storage units from the loading zone to rack and viceversa. Some other parts of the case study are not covered in the Figure 6.1 or are too small to be recognisable in it but they are essential parts of the WMS. For example, we can only see the loading platform in the yard but it consists of a gateway where the trucks register when they enter and deregister when they leave it. Inside the warehouse there are cameras used to monitor the automated retrieval and storage of the storage units (bins). We need these monitoring capabilities in order to

Figure 6.1: Warehouse from an external point of view.

have an overview of the actual warehouse status and to quickly react during runtime on several failure situations. This capability is very important in order to avoid down times of the system: a quick reaction to unusual situations is crucial to the system's availability.
The Figure 6.2 is a schematic representation of the main components in the warehouse. Thus, we can divide the whole warehouse management in three major parts: loading bay management, warehouse management system and yard management.

6.1.1 Main Components

In this section we will describe the details of the three major elements in the warehouse highlighted before.

loading bay management The loading bay is composed of several loading/unloading docks typically located at different parts of the warehouse where the trucks can go there and process their job. All the bins are carried on the belt and go to the warehouse incoming stack (in case of an incoming job) or come from the outgoing stack, then are placed on the belt and are carried in the truck (in case of outgoing job).

Warehouse management system If the bins arrive in the incoming stack,

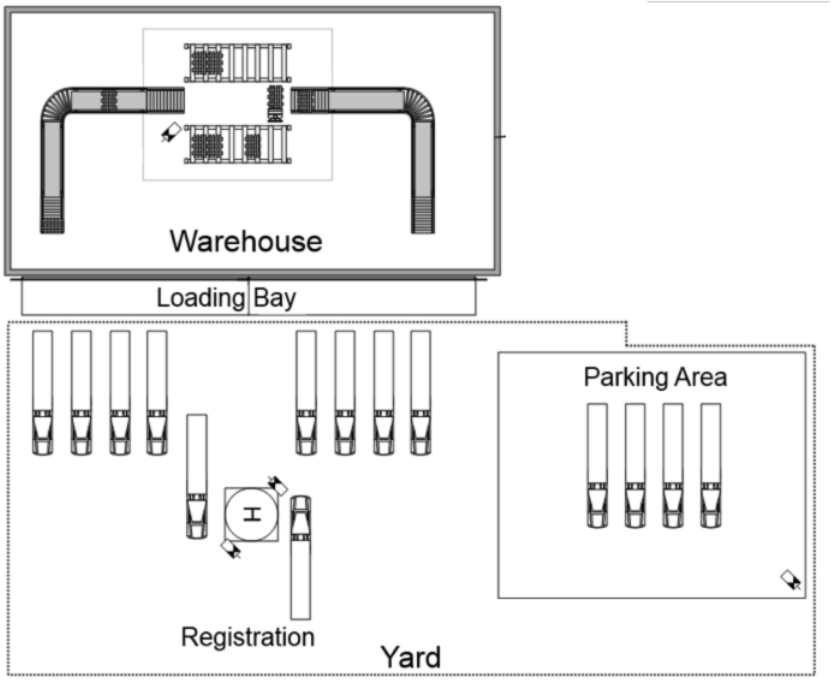

Figure 6.2: Warehouse from a schematic point of view.

left side of the warehouse in Figure 6.2, they are carried via automated stapler cranes to the pre-allocated rack by the warehouse management system via conveyers. The bins storage result is reported to the Enterprise Resource Planning system (ERP) that handles the overall commissioning of bins. In case of outgoing jobs, the bins are commissioned by the ERP system for delivery and the automated stapler crane takes the bins from the high rack to the outgoing stack. Afterwards, the bins are automatically placed on the conveyor which carries them to the loading issue dock.

yard The yard consists of a gateway where the trucks can register when they enter and deregister when they leave the yard to be scheduled and coordinated during the load and unload operations. The scheduling can be planned in advance with the help of shipping notifications of transport providers. Especially for large warehouses a parking place is needed to be able to handle a large amount of trucks especially during peak hours where a lot of trucks arrive and leave within a short period of time. Cameras are used to observe the trucks and the activities on the yard and the yard personnel are coordinated efficiently to run all processes smoothly.

Case Study UML Class Diagrams

The UML class diagram in Figure 6.3 shows the class that implements the adapter used in the monitored service to communicate with our IMF Runtime platform. In order to have a graphical view of what is happening in the warehouse during the execution, we developed a GUI and the other classes shown in the UML class diagram implement that functionality.

The UML class diagrams in Figure 6.4 and Figure 6.5 show our case study

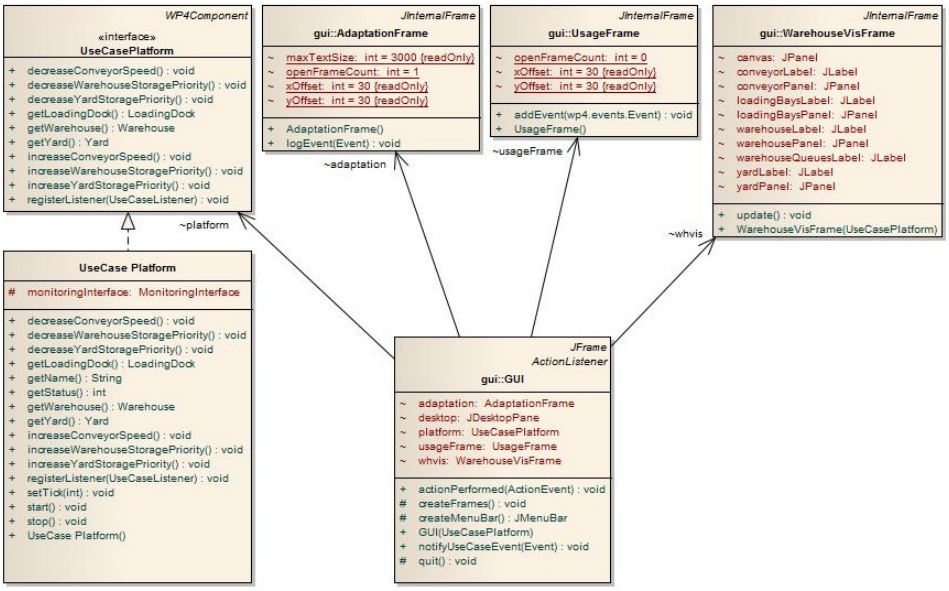

Figure 6.3: UML class diagram of the Case Study main platform and GUI.

in detail. In the first class diagram there are all the main classes used to implement the warehouse management system prototype. In the second class diagram there are all the main classes used to implement the yard management system and the loading bay prototype.

UML Class Diagram Case Study integration with the IMF Platform

In Figure 6.6 we report an exemplary UML sequence diagram. It shows the components initialization and an example of processing the receiving of some events about the state of the warehouse monitored service (*UseCase Platform*). In particular, the Adaptation Engine detects that the conveyor speed has to be increased so the Adaptation Interface calls the right method on the warehouse monitored service adapter to execute that adaptation action.

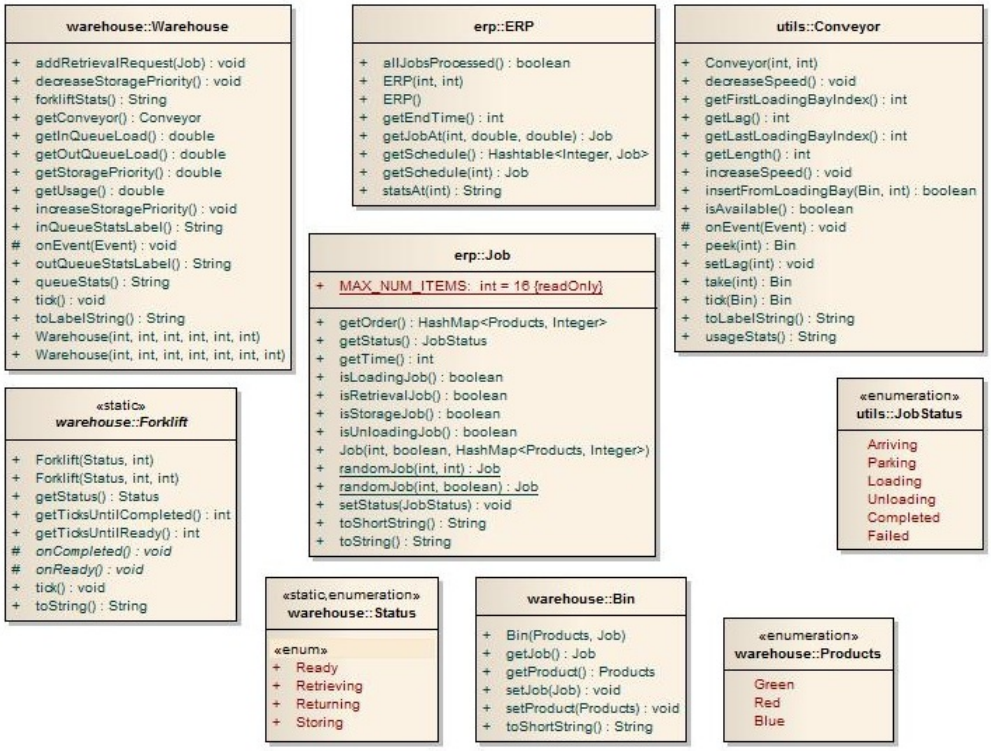

Figure 6.4: UML class diagram of the warehouse component.

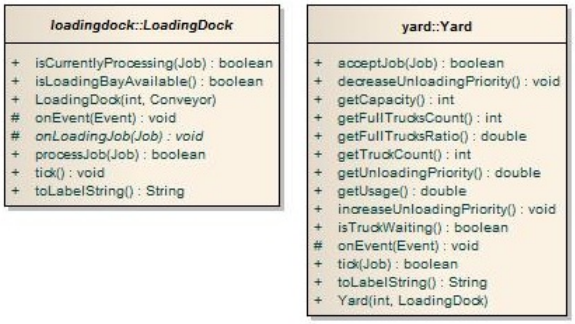

Figure 6.5: UML class diagram of the yard and loading dock components.

6.2 Performance

This section presents results of performance measurements of our IMF platform. As testbed we have our WMS case study scenario in the following organisation: the loading bay management, the warehouse management sys-

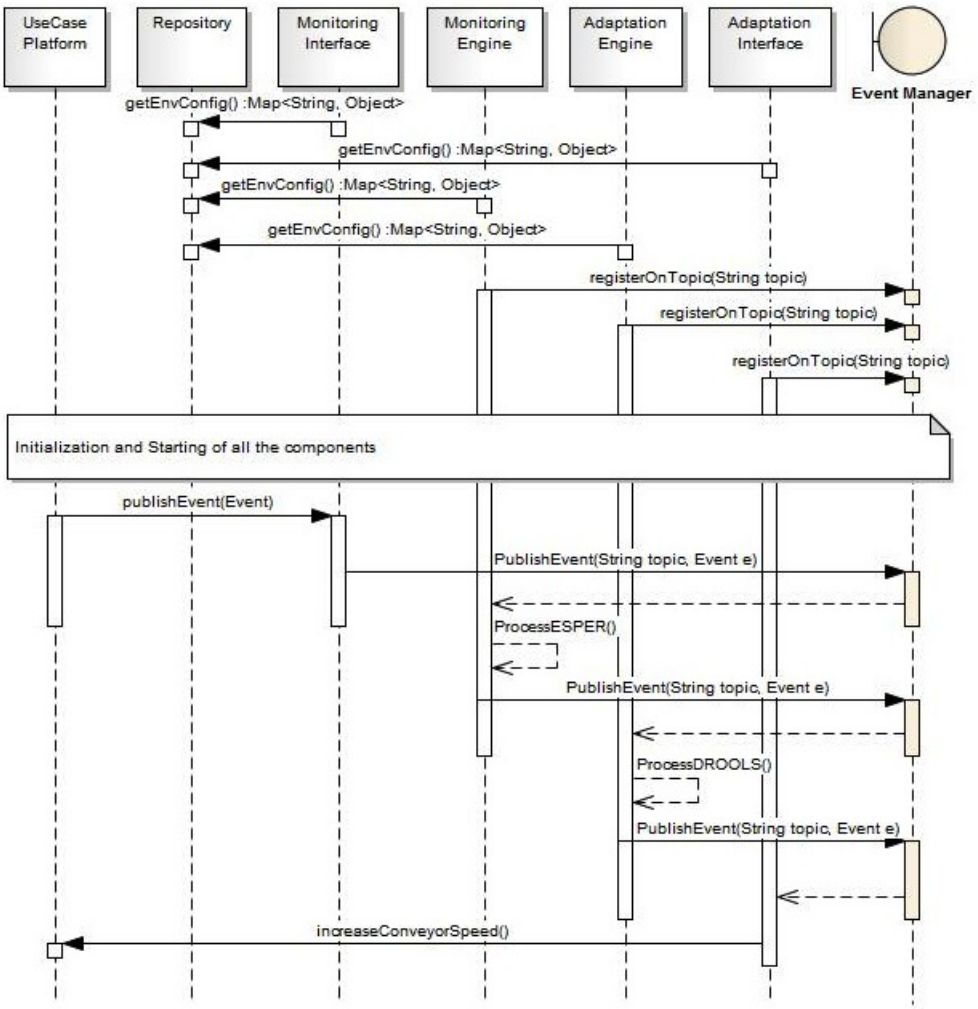

Figure 6.6: Exemplary UML sequence diagram.

tem and the yard management system are in different places of the whole warehouse and they have an adapter service running locally in order to communicate with our platform. In addition, we have our IMF Runtime platform running in the Internet. We don't know exactly the place where it is running, for example, we say that is running in the *Cloud*.
In order to evaluate the IMF performance, we need to define some policy rules. These rules are expressed in DROOLS language and are executed by the Adaptation Engine. These rules are:

- Decrease or increase the warehouse usage priority or set it to default value.
- Give priority to trucks in the yard to move to the loading docks with

incoming or outgoing jobs.

- Decrease or increase the conveyor speed.

For details and explanations about these policy rules, please refer to Appendix C. In order to get all the information needed by these policies, we defined some ESPER rules that are executed by the Monitoring Engine:

- warehouse usage
- outgoing and incoming stack usage
- parking area loading
- trucks with outgoing jobs vs incoming jobs ratio in the parking area

These rules are better explained in Appendix B.
We run our IMF Runtime platform in two main different scenarios in order to evaluate its performance, to show its flexibility and to find the best execution scenario for this case study. Before introducing the scenarios we will describe the techniques used to evaluate the performance and which was the event load for every single event type during the execution.
By analysing the monitoring rules, the generated monitoring events as input for the adaptation are five: *Usage*, *InStack*, *OutStack*, *YardFullTracksRatio* and *YardLoad*. The first three are coming from the Warehouse monitoring service and the last two from the Yard monitoring service. There are no events coming from the Loading Bay. By having a look on the adaptation rules, these events are used mostly for global adaptation and not local adaptation. This means that in most of the cases, the adaptation engine needs events from a monitored service and will apply business policies in another monitored service. Moreover, there are two adaptation rules that use composition of events coming from different monitored services. The only locally used event is the *Usage* event, all the others will fire adaptation actions on monitored services different from the one that generated those events. Thus, only 20% of the monitoring events load could be managed by a local execution of the IMF Runtime platform and 25% of the business rules has to be absolutely executed by an IMF Runtime platform in the Cloud. In order to better analyse all the consequences, we will consider latency and CPU usage as the major parameters. For the latency formula we will sum the packet round-trip time and the execution time. We can ignore the round-trip time for local messages but the round-trip time for messages to the Cloud is relevant. Moreover, if we use a Gateway component, we have to add some processing time to the round-trip time. The IMF Runtime platform on the

Cloud can obviously execute more rules per second than the local IMF Runtime platform and a possible CPU usage formula will be fired rules divided by maximum executable rules per second.

6.2.1 Basic Scenario

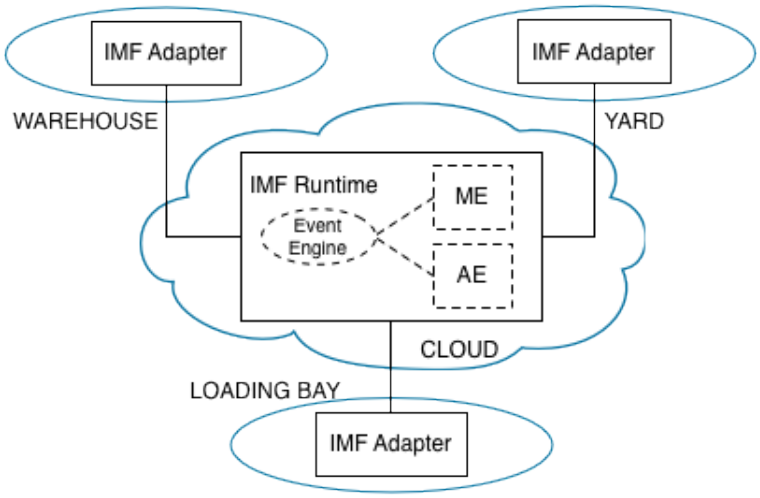

Figure 6.7: Case Study Basic Scenario.

As shown in Figure 6.7, the basic scenario is realised by hosting the IMF Runtime platform in the Cloud. Every single monitored service has an adapter in order to communicate with the IMF Runtime platform Interfaces. In this scenario all the monitoring rules and adaptation rules are executed in the Cloud. By considering our performance indicators, we soon understand that this is not the best scenario and the results are the proof. In particular, the latency time for every fired monitoring event from the monitored service is more then ten seconds and the CPU in the Cloud will be busy in most of the time. Moreover, this solution doesn't support all the features of our IMF Runtime platform thus, events like *Usage* will be sent in the Cloud and will come back in the same platform.

6.2.2 Advanced Scenario

As shown in Figure 6.8, the advanced scenario is realised by hosting an IMF Runtime platform locally in every single service platform that we want to monitor and a global IMF Runtime platform in the Cloud. This scenario has a new component that enables the catching of local events that need to be processed by the global IMF Runtime platform in the Cloud: the

Gateway. We need the Gateway component because the IMF Tools enables the user to define some policy rules that need to evaluate events coming from different monitored services. In particular, the Gateway component will register to the local Event Manager and publish the events in the Event Manager running in the Cloud. This solution will reduce the CPU usage in the Cloud because, for example, the *Usage* event will never be sent out of the Warehouse management system boundaries. As result, we will have a better performance in the adaptation of the warehouse, a consequence on which we have to pay attention because we don't want the warehouse to be full. Filling the whole warehouse will lead to a deadlock in the whole use case so we want an autonomic control as much as possible close to the realtime. In this scenario we show also that our IMF Runtime platform is flexible and we apply two different hierarchical levels of autonomic management. Another solution that can be derived from this scenario and that is the best one from our point of view is running only an adapter component on the Loading Bay service and this will remove the overhead of the Gateway.

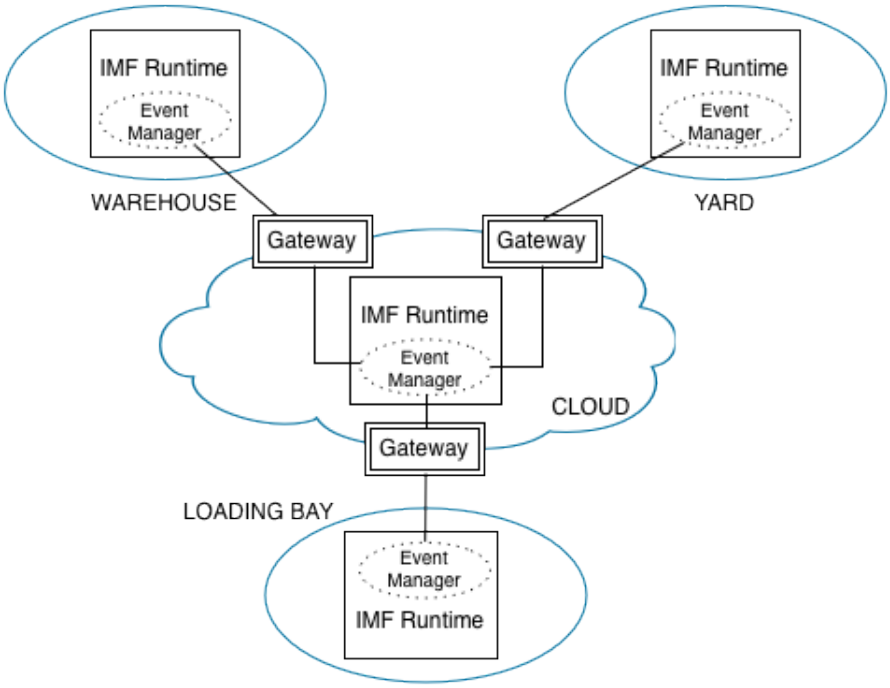

Figure 6.8: Case Study Advanced Scenario.

Conclusions

The target of this thesis work was developing a framework for heterogeneous systems self-adaptation. Actual companies scenario is characterised by several heterogeneous systems and the first step is trying to integrate them together. Moreover, wishing to manage all these systems together is out of human possibilities. Another important aspect is that these systems cannot be considered in a closed world but we are in a open environment characterised by dynamic evolution that the stakeholder often can't predict. This is why several research studies were done about how could be possible to automatically manage these heterogeneous systems. These studies result is that the component-based development is a possible way to communicate with these heterogeneous systems running in an open-world. The main actor in the development process will be the Virtual Service Platform.
We developed INDENICA Management Framework, a Virtual Service Platform that is able to manage these systems. We can find several solutions in the market in order to implement service components. Our solution used to let the monitored system and our IMF Runtime Platform is implement service component in Apache Tuscany for Java. Another important aspect to consider about these Virtual Service Platforms is how the internal components can communicate without loosing flexibility. We decided to implement a communication based on the publish-subscribe design with RabbitMQ. We also developed the IMF tool, a framework that helps to configure the IMF Platform and enable the customers to define their business rules in well know languages by using the Esper CEP engine for the events processing and the JBoss Drools engine to define the policies. We also provided and developed a case study based on a warehouse management. This case study reveals all the features of our INDENICA Management Framework but also its performance. The service-oriented technology seems to be capable of dealing with the matter, since the solution has been tested in a case study and in the future we will test it in another real case study.

The INDENICA Management Framework is still in a prototype state and we are thinking about extending some functionality. Currently, the monitoring rules are provided manually but in a future version of the IMF tool it could integrate a view-based modelling approach that will enable to automatically generate monitoring rules from UML models and build up a library of reusable monitoring rules. The same development work can affect the adaptation side of the IMF tool. The Monitoring Engine instances and the Adaptation Engine instances are defined in a textual way, a future work can be that of providing a more visually appealing interface for defining components and their interaction using a drag-and-drop approach. The IMF Platform is now able to automatically adapt itself by defining some higher level business rules but the automatic scalability by passing rules between engines is a missing feature. Actually, every Engine has its own rules and they cannot change at runtime but this is a feature that could be extended in the future.

Bibliography

[1] Apache. Apache tuscany. http://tuscany.apache.org/.

[2] Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, and Kishore Channabasavaiah. S3: A service-oriented reference architecture. *IT Professional*, 9(3):10–17, 2007.

[3] Luciano Baresi, Neil Maiden, and Peter Sawyer. Service-centric systems and requirements engineering. *Requirements Engineering, IEEE International Conference on*, 0:305, 2008.

[4] Thomas Bernhardt and Alexandre Vasseur. Esper: Event stream processing and correlation, 2007.

[5] Thomas Bernhardt and Alexandre Vasseur. Complex event processing made simple using esper, 2008.

[6] Sami Beydeda and Volker Gruhn. *Model-Driven Software Development*. Springer-Verlag, 2005.

[7] Aliaksandr Birukou, Vincenzo D'Andrea, Frank Leymann, Jacek Serafinski, Patrícia Silveira, Steve Strauch, and Marek Tluczek. An integrated solution for runtime compliance governance in SOA. In *Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC'10), San Francisco, California, USA, December 7-10, 2010*, Artikel in Tagungsband, pages 122–136. Springer, December 7, 2010.

[8] Ron Burback. A distributed architecture definition language: a DADL. http://citeseer.ist.psu.edu/435593.html; http://www-db.stanford.edu/ burback/dadl/dadl.ps, 1998.

[9] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti, and Raffaela Mirandola. Qos-driven runtime adaptation of service oriented architectures. In Hans van Vliet and Valérie Issarny, editors, *Proceedings of the 7th joint meeting of the European Software*

Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009, Amsterdam, The Netherlands, August 24-28, 2009, pages 131–140. ACM, 2009.

[10] David Chappell. Introducing sca, 2007.

[11] Kristina Chodorow and Michael Dirolf. *MongoDB - The Definitive Guide: Powerful and Scalable Data Storage.* O'Reilly, 2010.

[12] Open SOA Consortium. Service component architecture (sca). http://www.oasis-opencsa.org/sca.

[13] World Wide Web Consortium. Web services architecture, 2004. http://www.w3.org/TR/ws-arch/.

[14] Ivica Crnkovic, Stig Larsson, and Michel R. V. Chaudron. Component-based development process and component lifecycle. *CIT*, 13(4):321–327, 2005.

[15] Luiz Olavo Bonino da Silva Santos, Fano Ramparany, Patricia Dockhorn Costa, Peter Vink, Richard Etter, and Tom Broens. A service architecture for context awareness and reaction provisioning. In *IEEE SCW*, pages 25–32. IEEE Computer Society, 2007.

[16] Luiz Olavo Bonino da Silva Santos, Remco Poortinga van Wijnen, and Peter Vink. A service-oriented middleware for context-aware applications. In Sotirios Terzis, Steve Neely, and Nitya Narasimhan, editors, *Proceedings of the 5th International Workshop on Middleware for Pervasive and Ad-hoc Computing (MPAC 2007), held at the ACM/IFIP/USENIX 8th International Middleware Conference, November 26 - November 30, 2007, Newport Beach, California, USA*, ACM International Conference Proceeding Series, pages 37–42. ACM, 2007.

[17] Robert Daigneau. *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services.* Addison-Wesley Professional, 1 edition, November 2011.

[18] Premkumar Devanbu and Eric Wohlstadter. Evolution in distributed heterogeneous systems, 2001. Premkumar Devanbu (Dept . Of Computer Science,; University of California; Davis , CA , 95616); Eric Wohlstadter (Dept . Of Computer Science,; University of California; Davis , CA , 95616);.

[19] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[20] EsperTech. EsperTech: Esper, event driven intelligence. http://esper.codehaus.org/.

[21] Eugster, Felber, Guerraoui, and Kermarrec. The many faces of publish/subscribe. *CSURV: Computing Surveys*, 35, 2003.

[22] Ludger Fiege, Felix C. Gärtner, Oliver Kasten, and Andreas Zeidler. Supporting mobility in content-based publish/subscribe middleware. In Markus Endler and Douglas C. Schmidt, editors, *Middleware 2003, ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, June 16-20, 2003, Proceedings*, volume 2672 of *Lecture Notes in Computer Science*, pages 103–122. Springer, 2003.

[23] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of Califormia, Irvine, USA, 2000.

[24] C. L. Forgy. Rete: A fast algorithm for the many pattern / many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[25] Apache Software Foundation. Apache licence, version 2.0. http://www.apache.org/licenses/LICENSE-2.0.

[26] Robert B. France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In Lionel C. Briand and Alexander L. Wolf, editors, *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA*, pages 37–54, 2007.

[27] Eli Gjørven, Frank Eliassen, and Romain Rouvoy. Experiences from developing a component technology agnostic adaptation framework. In Michel R. V. Chaudron, Clemens A. Szyperski, and Ralf Reussner, editors, *Component-Based Software Engineering, 11th International Symposium, CBSE 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings*, volume 5282 of *Lecture Notes in Computer Science*, pages 230–245. Springer, 2008.

[28] Eli Gjørven, Romain Rouvoy, and Frank Eliassen. Cross-layer self-adaptation of service-oriented architectures. In Karl M. Göschka, Schahram Dustdar, Frank Leymann, and Vladimir Tosic, editors, *Proceedings of the 3rd Workshop on Middleware for Service Oriented Computing, MW4SOC 2008, Leuven, Belgium, December 1-5, 2008*, pages 37–42. ACM, 2008.

[29] W3C Hugo Haas. Designing the architecture for web services. http://www.w3.org/2003/Talks/0521-hh-wsa/.

[30] INDENICA. INDENICA: research project co-founded by the european commission within the 7th framework programme in the area internet of services, software & virtualisation (ict-2009.1.2). http://www.indenica.eu/.

[31] JBoss. JBoss Drools: Business logic integration platform. http://www.jboss.org/drools/.

[32] Jeff Johnson John Plummer. Complex event processing, 2008.

[33] Diane Jordan and John Evdemon. Web services business process execution language version 2.0. OASIS Standard, April 2007.

[34] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36:41–50, January 2003.

[35] Jaroslaw Kijanowski. Introduction to drools: Rules fall from your eyes. *RED Hat Magazine*, 07, 2008.

[36] David C. Luckham. *The power of events - an introduction to complex event processing in distributed enterprise systems.* ACM, 2005.

[37] Haleh Mahbod, Raymond Feng, and Simon Laws. What is sca? a quick view of concepts through and an example walkthrough. *Java Developer Journal*, Feb 2007.

[38] Qusay H. Mahmoud. Getting started with the java rule engine api (jsr 94): Toward rule-based applications, 2005.

[39] Mongo. Mongodb: name:mongo type:db. http://www.mongodb.org/.

[40] Christoph Nagl, Florian Rosenberg, and Schahram Dustdar. VIDRE - A distributed service-oriented business rule engine based on ruleML. In *EDOC*, pages 35–44. IEEE Computer Society, 2006.

[41] The Object Management Group (OMG). Omg mda guide. version 1.0.1. http://www.omg.org/, 2003.

[42] Web Services Interoperability Organisation. Web services interoperability organisation. http://www.ws-i.org/deliverables/.

[43] M. Oriol Hilari, J. Marco Gómez, J. Franch Gutiérrez, D. Ameller, et al. Monitoring adaptable soa systems using salmon. *Workshop in ServiceWave conference, Madrid*, 2008.

[44] Mark Palmenr and Michal Dzmuran. An introduction to event processing. PROGRESS software, 2007.

[45] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: A research roadmap. *Intermational Journal of Cooperative Information Systems*, 17(2):223–255, 2008.

[46] F. Ramparany, R. Poortinga, M. Stikic, J. Schmalenströer, and T. Prante. An open context information management infrastructure the IST-amigo project. *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on Intelligent Environments*, pages 398–403, September 2007.

[47] RuleML. The rule markup initiative, 2005. http:// www.ruleml.org/, 2005.

[48] Benjamin Satzger, Andreas Pietzowski, Wolfgang Trumler, and Theo Ungerer. Using automated planning for trusted self-organising organic computing systems. In *5th International Conference on Autonomic and Trusted Computing (ATC 2008)*, pages 60–72, 2008.

[49] R. Soley, D. Frankel, J. Mukerji, and E. Castain. Model driven architecture - the architecture of choice for a changing world. Technical report, OMG, 2001. http://www.omg.org/.

[50] MediaLab Sonera Plaza Ltd. Web services white paper. http://www.medialab.sonera.fi/, 2002.

[51] Metafrom Systems. Fabric3. http://www.fabric3.org/.

[52] Yongzhong Tang and Baotai Liu. Design high reliable monitor and control system using event-driven soa philosophy. *2009 IEEE International Symposium on IT in Medicine Education*, pages 146–153, 2009.

[53] JBoss Drools Team. Drools expert user guide. *JBoss Community Documentation*, 2010.

[54] S. Vinoski. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 35(2):46–55, 1997.

[55] VMware. Rabbitmq: messaging that just works. http://www.rabbitmq.com/.

[56] WebServices.org. WebServices.org: serving the soa community since 2001. http://www.webservices.org/.

[57] Krzysztof Zielinnski, Tomasz Szydlo, Robert Szymacha, Jacek Kosinski, Joanna Kosinska, and Marcin Jarzab. Adaptive soa solution stack. *IEEE Transactions on Services Computing*, 99(PrePrints), 2011.

[58] Daniel Zmuda, Marek Psiuk, and Krzysztof Zielinski. Dynamic monitoring framework for the SOA execution environment. *Procedia CS*, 1(1):125–133, 2010.

Appendix A

INDENICA Management Platform User Manual

A step-by-step guide to creating a basic INDENICA Management Framework Runtime instance using the Repository and the provided tools.

A.1 Usage Guide

In this section we provide a step-by-step guide for creating a basic instance of the IMF Runtime infrastructure using the IMF Tools.
In order to create a new instance of the IMF Runtime infrastructure, running instances of RabbitMQ and MongoDB are required. To guide the user through the steps necessary to create a runtime infrastructure instance, a simple launch dialog can be invoked using:

```
sh launchConfiguration.sh
```

In Figure A.1 is shown the main configuration launch dialog.

Next, a connection to the repository database must be established by invoking "File → Connect to DB" (Figure A.2).

The data entered in the connection dialog must point to a running MongoDB instance in order to complete correctly (Figure A.3).

After successfully establishing a connection to the database, an environment configuration for a new infrastructure instance can be created by selecting "Environment Configurator" in the dropdown list and clicking "Launch" (Figure A.4).

To create a new environment configuration, select "File → New Configuration" (Figure A.5).

Must be provided a name for the infrastructure instance as shown in the

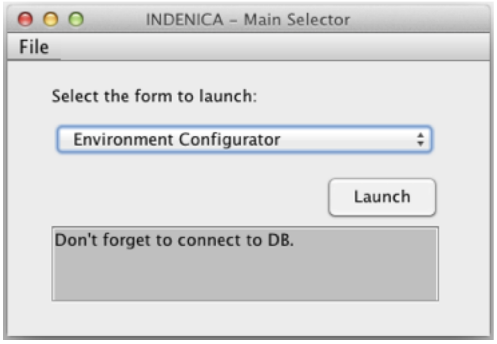

Figure A.1: Configuration Launch Dialog.

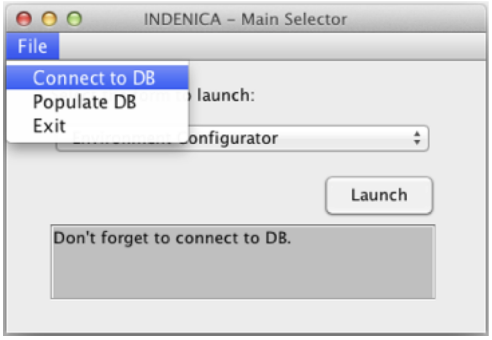

Figure A.2: Connect to Repository Database.

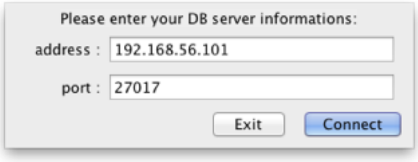

Figure A.3: Establish Database Connection.

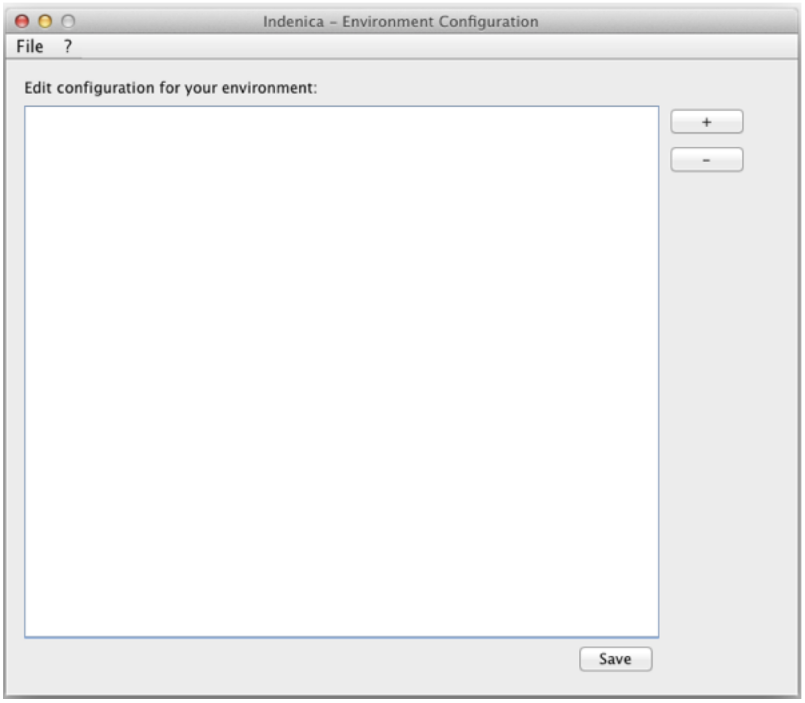

Figure A.4: Environment Configuration Dialog.

Figure A.5: Create new Environment Configuration.

dialog of Figure A.6 (in our case, "SampleInstance").

After confirming the creation of the new instance configuration, the environ-

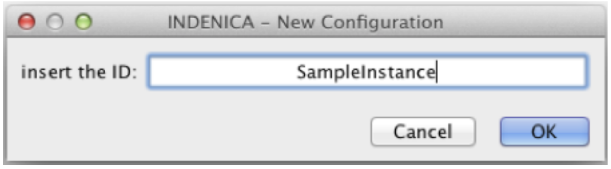

Figure A.6: New Environment Configuration Dialog.

ment configuration is pre-populated with relevant configuration properties that need to be provided by the user. Where appropriate, the values are set to factory default settings, or contain usage information (Figure A.7).

Figure A.8 shows an exemplary configuration for an infrastructure instance.

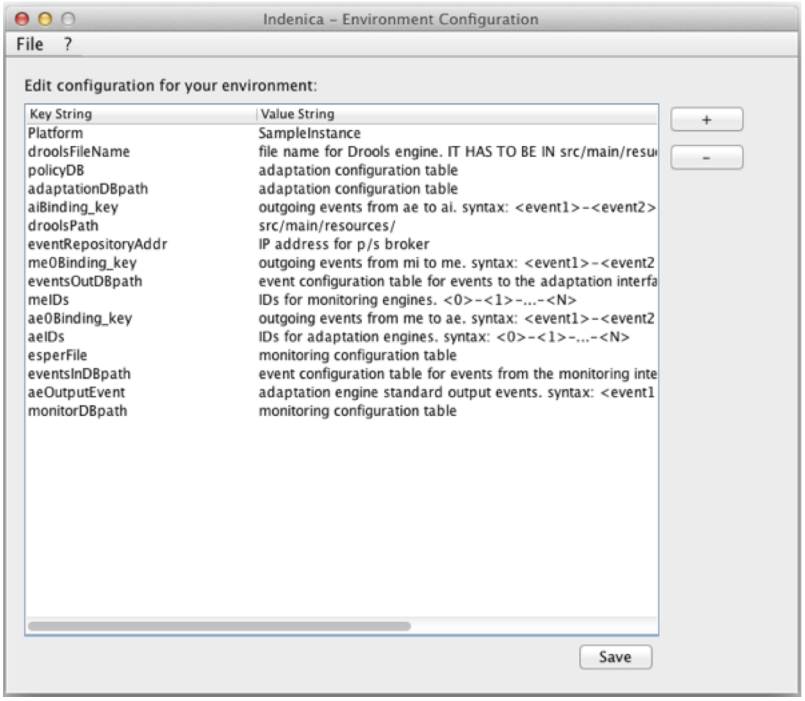

Figure A.7: Environment Configuration Defaults and Usage Hints.

This configuration creates one Monitoring Engine (ME) instance, as well as one Adaptation Engine (AE) instance and defines events that these components receive.

The configuration can then be saved by using the "Save" button on the lower right and the "Environment Configuration" dialog can be closed.

The next step in setting up an infrastructure instance involves defining con-

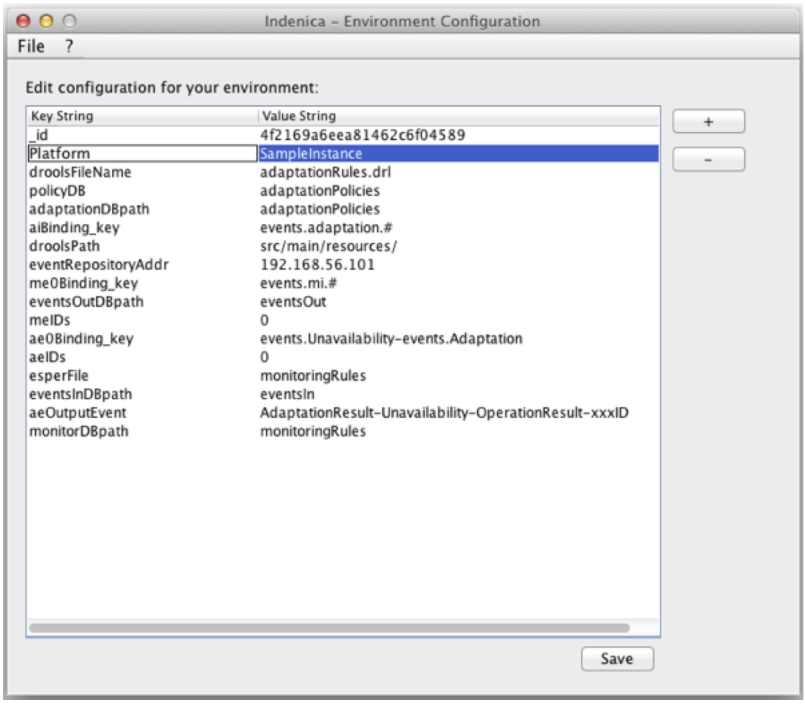

Figure A.8: Sample Infrastructure Instance Environment Configuration.

crete events that the monitoring interfaces emit for consumption by the previously defined monitoring engine.
In the launcher dialog, select "Events Incoming" in the dropdown box and click "Launch" as shown in Figure A.9.

The previously created environment configuration must be loaded in the

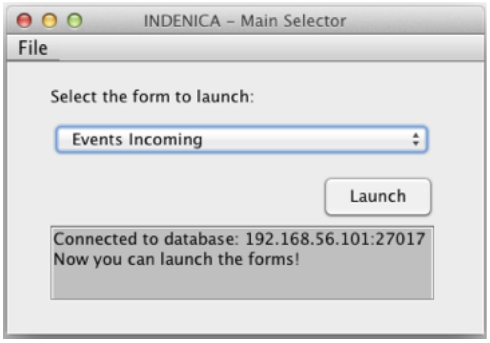

Figure A.9: Launch Incoming Events Dialog.

incoming events dialog (Figure A.10).

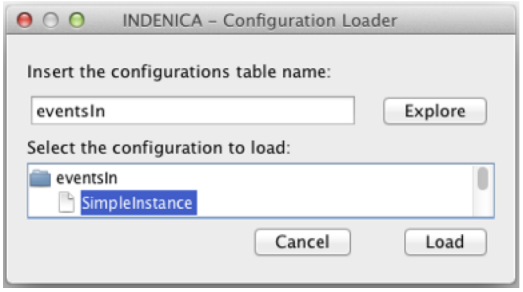

Figure A.10: Load previously created Configuration.

To load an environment configuration, select "File → Load Configuration" click "Explore" and select "SimpleInstance" (Figure A.11).

Now, events originating from the monitoring interface can be created. Figure A.12 shows an exemplary event type, 'ServiceInvocationEvent', and according properties.

Event types and their settings are saved using the "Add Event" button on the bottom of the dialog. For the sample instance, we will add a second event type, 'ServiceInvocationFailureEvent' as depicted in Figure A.13.

The next step involves specifying adaptation events sent to the adaptation interface in order to influence integrated service platforms. Selecting "Events Outgoing" in the launch dialog opens the according dialog (Figure A.14).

Again, the environment configuration must be loaded by invoking "File → Load Configuration", clicking "Explore" and selecting "SampleInstance". Events sent to the adaptation interface can be created similar to the incoming events. For our sample instance, we will create two events representing a notification about a healthy or unhealthy system state ("adaptation.SystemOK" and "adaptation.SystemKO"), as shown in the two Figures A.15 and A.16.

After saving the outgoing adaptation interface events, we can now configure the monitoring engine by selecting the according option in the dropdown menu in the launch dialog (Figure A.17).

After loading the environment configuration by invoking "File → Load Configuration", clicking "Explore" and selecting "SimpleInstance", the specified Monitoring Engines can be configured.

To configure *Monitoring Engine 0*, we select it in the "Engine ID" dropdown and choose "Load". Now, monitoring rules can be added, using the rules specified in the "Events Incoming" dialog. The current implementation supports rules in the Esper Query Language (EQL).

Figure A.18 shows an exemplary monitoring rule evaluating the availability ratio of a monitored service by analysing successful and failed invocations over a period of one day.

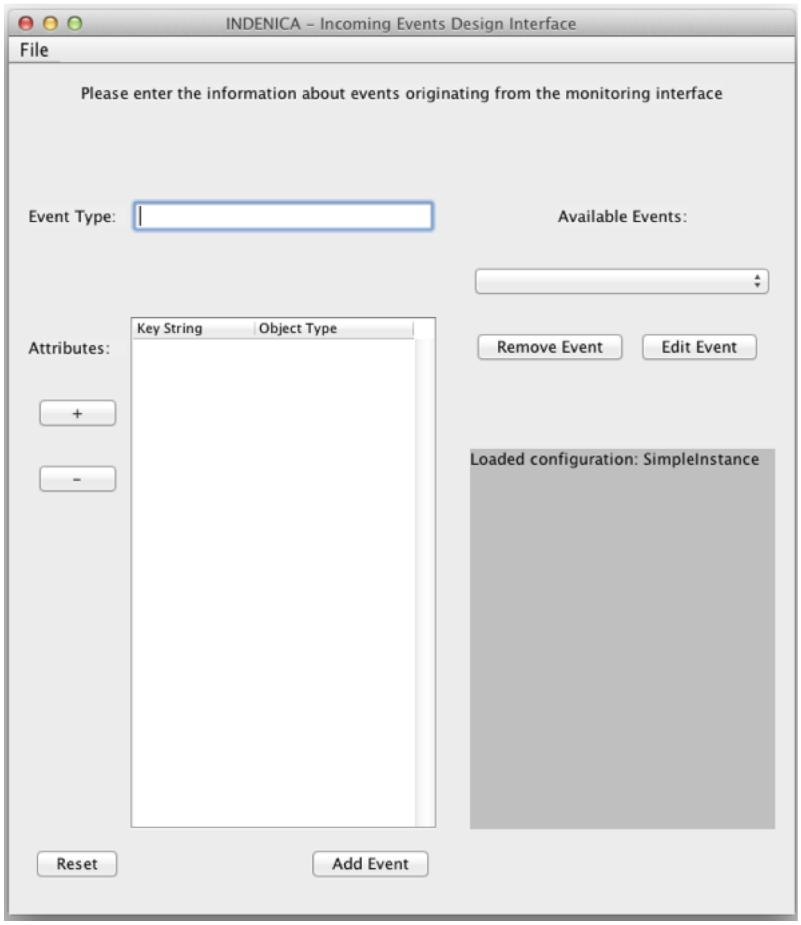

Figure A.11: Incoming Events Dialog.

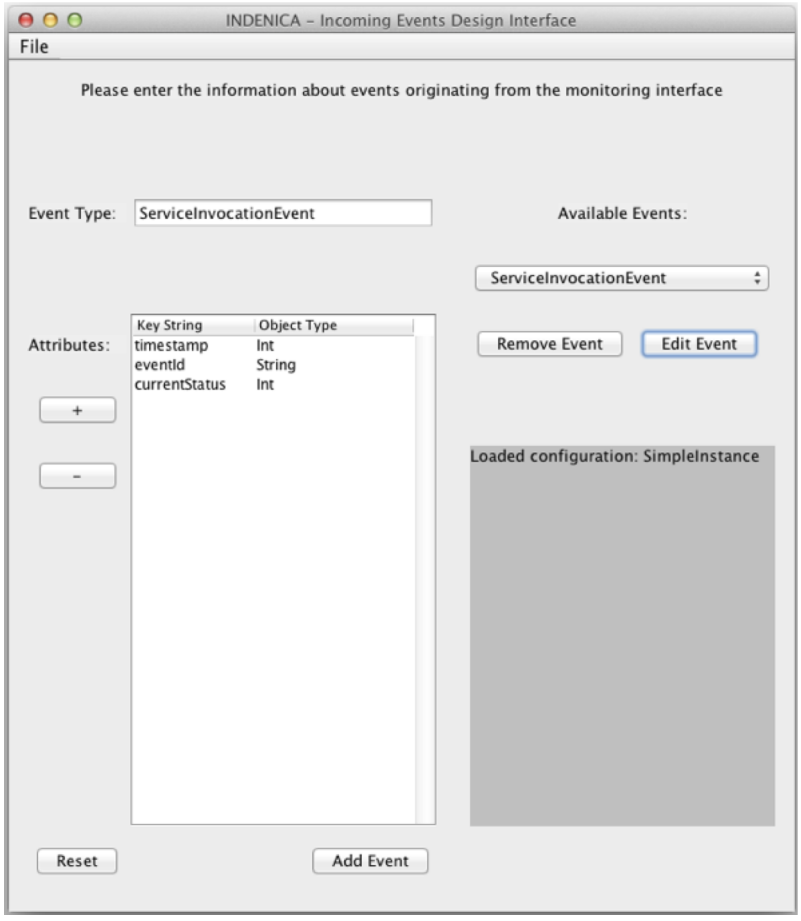

Figure A.12: An exemplary Monitoring Event Type.

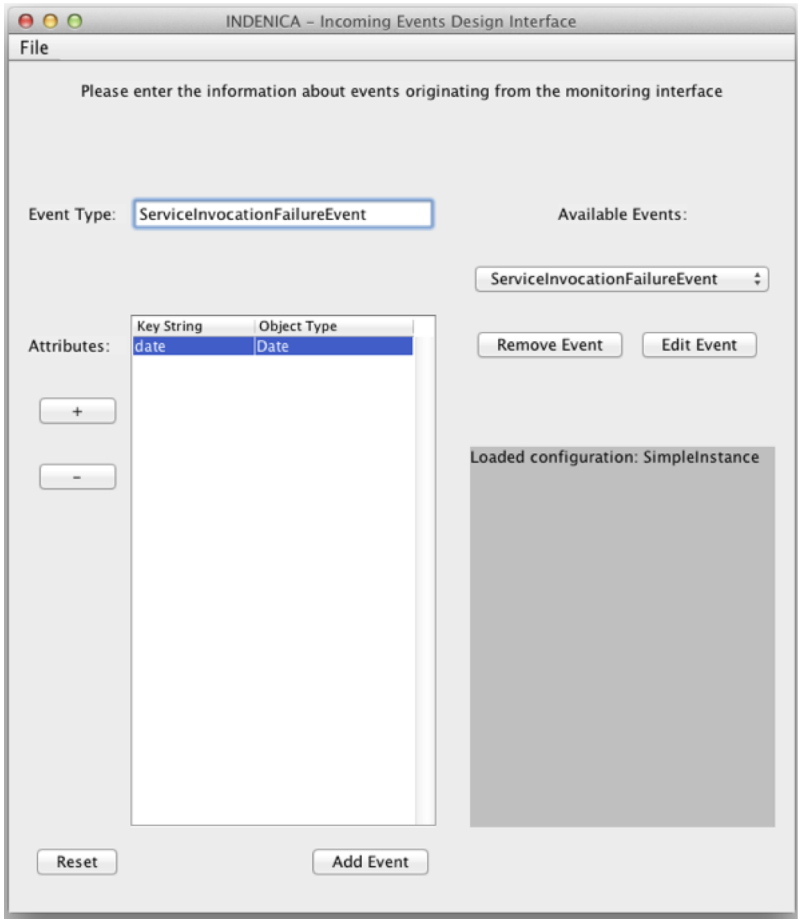

Figure A.13: ServiceInvocationFailure Event Type.

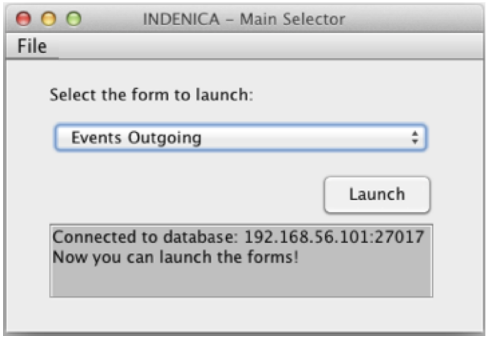

Figure A.14: Launch the outgoing Adaptation Events Dialog.

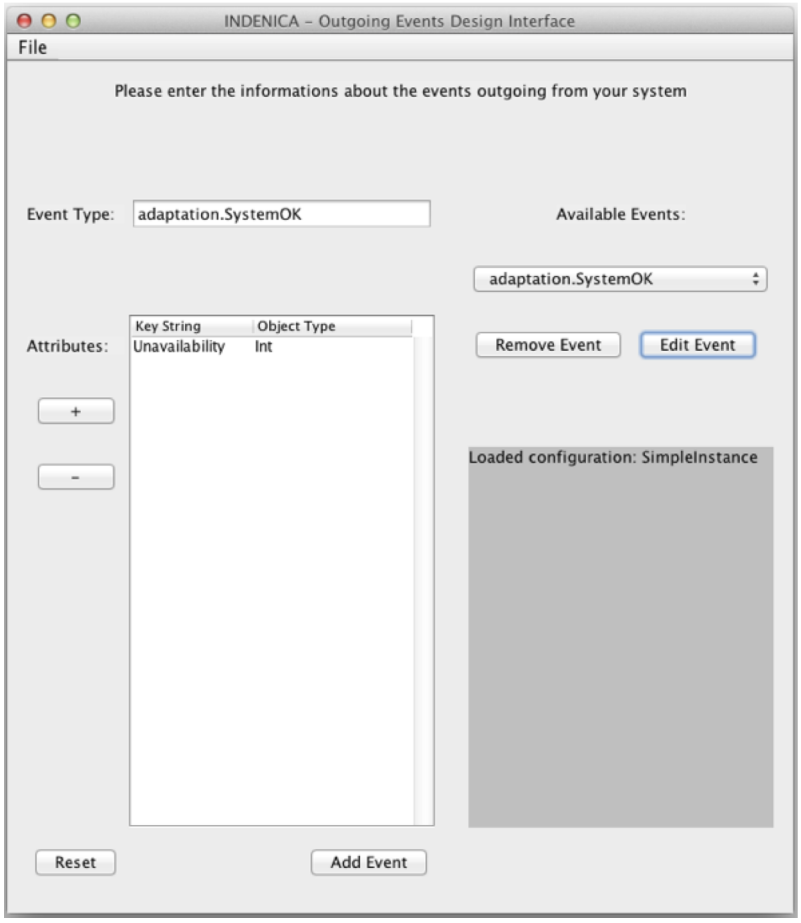

Figure A.15: Exemplary Adaptation Interface Event.

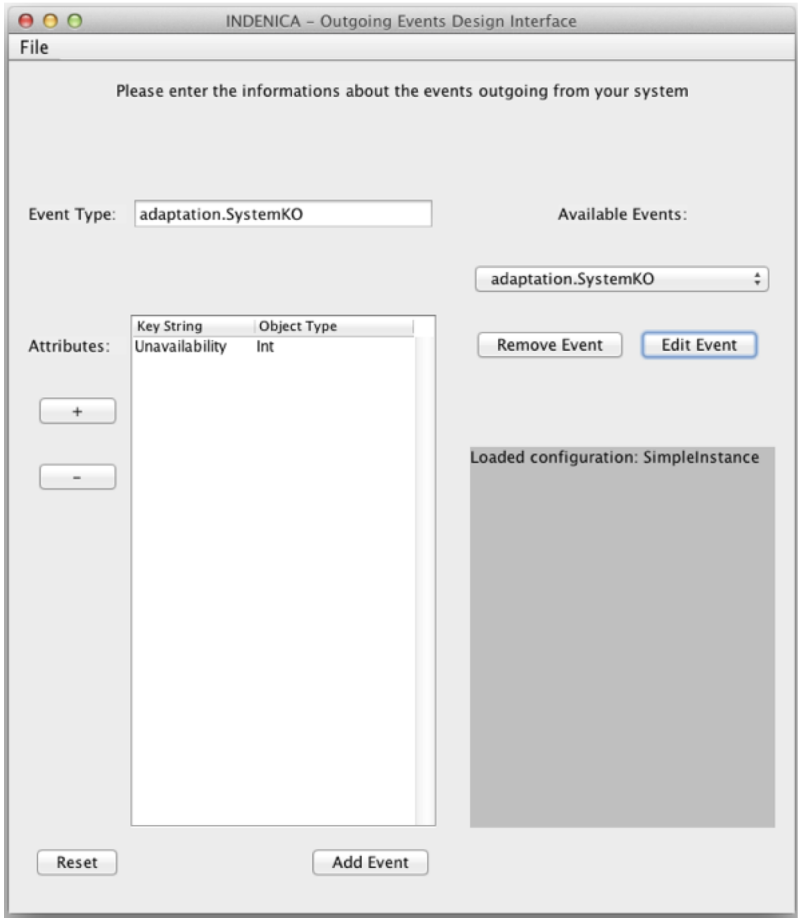

Figure A.16: Exemplary Adaptation Interface Event.

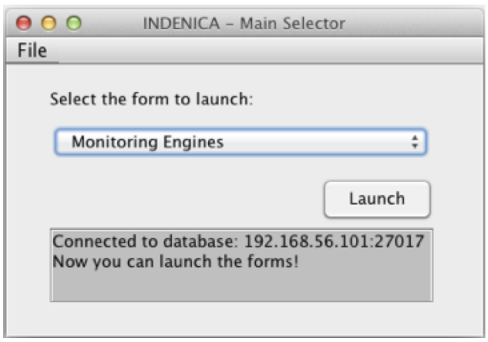

Figure A.17: Launch Monitoring Engine Configuration.

After saving the monitoring rule (using "Add Rule" on the lower right), we

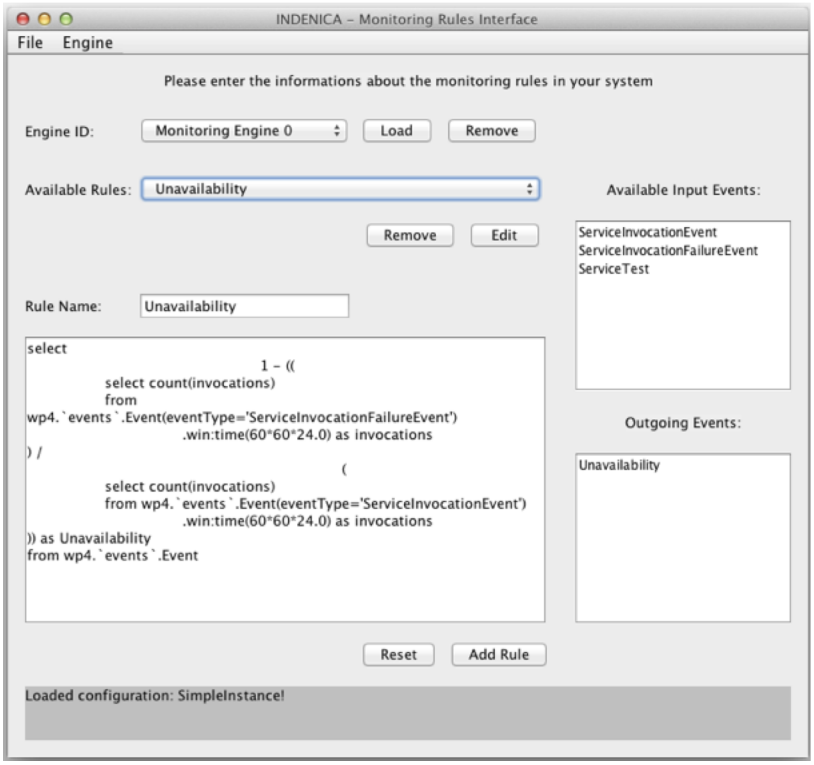

Figure A.18: Exemplary Monitoring Rule.

can configure actions the AE should take in response to certain monitoring events by invoking the "Adaptation Engines" configuration in the launch dialog (Figure A.19).

After loading the environment configuration by invoking "File → Load Configuration", clicking "Explore" and selecting "SimpleInstance", the specified AEs can be configured.

To configure *Adaptation Engine 0*, we select it in the "Engine ID" dropdown and choose "Load". Now, adaptation rules can be added, using the event specified in the ME configuration dialog. The current prototype implementation supports adaptation rules in the Drools rule language.

For the sample instance, we will create two simple adaptation rules, as shown in Figure A.20 and Figure A.21.

The adaptation rules complete the environment configuration. A platform integrator is now required to create the appropriate Monitoring Interface (MI) and Adaptation Interface (AI) for the integrated platforms to interact with the IMF Runtime platform. In the current implementation the infrastructure library is represented by java packages 'indenica.[monitoring |

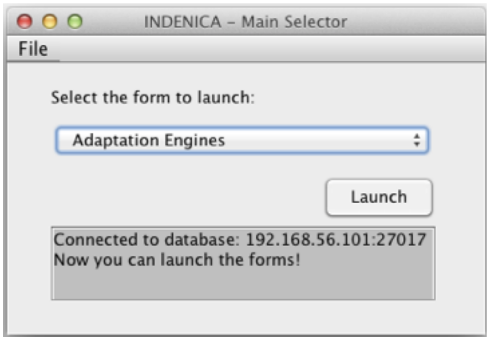

Figure A.19: Launch Adaptation Engine Configuration.

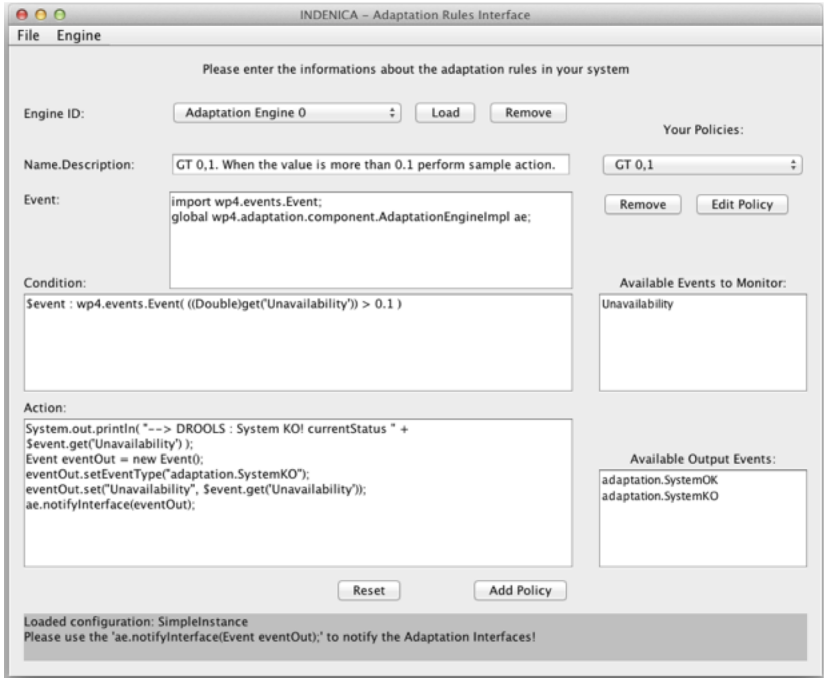

Figure A.20: Exemplary Adaptation Rule.

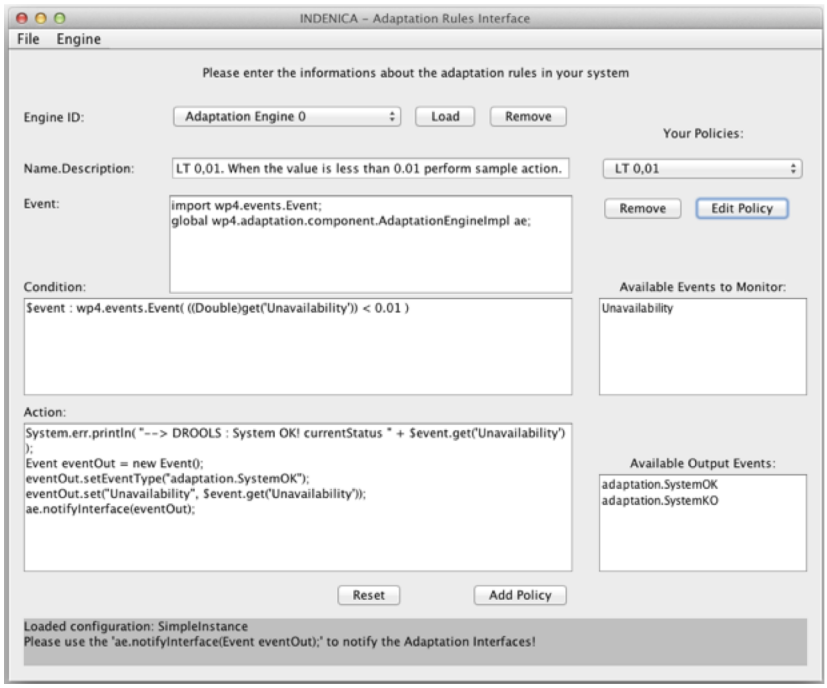

Figure A.21: Exemplary Adaptation Rule.

adaptation].component'.
To invoke the created platform, a Tuscany configuration composite file is needed. In the current implementation, this file has to be supplied manually.

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
   targetNamespace="http://indenica.eu/repository"
   xmlns:hw=" http://indenica.eu/repository" name="
   IndenicaRuntime">

  <component name="ComponentInitializerComponent">
    <implementation.java class="indenica.deployment.
       component.ComponentInitializerImpl" />
    <reference name="adaptationInterface" target="
       AdaptationInterfaceComponent" />
    <reference name="repository" target="
       RepositoryComponent" />
    <reference name="platform" target="
       SimplePlatformComponent" />
  </component>
```

```
  <component name="MonitoringInterfaceComponent">
    <implementation.java class="indenica.monitoring.
       component.SimpleMonitoringInterfaceImpl" />
  </component>

  <component name="AdaptationInterfaceComponent">
    <implementation.java class="indenica.adaptation.
       component.SimpleAdaptationInterfaceImpl" />
    <reference name="platform" target="
       SimplePlatformComponent" />
    <reference name="repository" target="
       RepositoryComponent" />
  </component>

  <component name="RepositoryComponent">
    <implementation.java class="indenica.repository.
       component.RepositoryImpl" />
    <property name="dbAddress">192.168.56.101</
       property>
    <property name="dbPort">27017</property>
    <property name="platform">SimplePlatform</
       property>
    <property name="adminDB">adminDB</property>
  </component>

  <component name="SimplePlatformComponent">
    <implementation.java class="indenica.sample.
       SimplePlatformImpl" />
    <reference name="monitoringInterface" target="
       MonitoringInterfaceComponent" />
  </component>
</composite>
```

After saving the composite file in the 'src/main/resources' directory, the completed infrastructure instance can now be started using:

```
java -jar indenicaInfrastructure.jar indenica.
   deployment.Launcher <composite file name>
```

Appendix B

Case Study ESPER Monitoring Rules

In this Appendix we will introduce and describe all the ESPER monitoring rules used in the Warehouse case study.

B.1 Warehouse

In the following listings we will describe all the monitoring rules used to monitor the Warehouse component.
The following listing receives events about the current load in the stack for incoming items in the warehouse. The ESPER engine will collect 10 events of type *warehouse.queues.in* and send an average of the *load* value as a new event named *InStack*.

```
select avg(cast(ev.`getValue`('load'), double)) as
   InStack
from indenica.`events`.Event(eventType='warehouse.
   queues.in').win:length_batch(10) as ev
where ev.`getValue`('itemAddedSuccessfully') = true
```

The following listing receives events about the current load in the stack for outgoing items in the warehouse. The ESPER engine will collect 10 events of type *warehouse.queues.out* and send an average of the *load* value as a new event named *OutStack*.

```
select avg(cast(ev.`getValue`('load'), double)) as
   OutStack
from indenica.`events`.Event(eventType='warehouse.
   queues.out').win:length_batch(10) as ev
where ev.`getValue`('itemAddedSuccessfully') = true
```

The following listing receives events about the current load in the warehouse. The ESPER engine will collect 10 events of type *warehouse.status* and send an average of the *usage* value as a new event named *Usage*.

```
select avg(cast(ev.`getValue`('usage'), double)) as
   Usage
from indenica.`events`.Event(eventType='warehouse.
   status').win:length_batch(10) as ev
```

B.2 Yard

In the following listings we will describe all the monitoring rules used to monitor the Yard component.
The following listing receives events about the current load in the yard parking zone. The ESPER engine will collect 10 events of type *yard.status* and send an average of the *load* value as a new event named *YardLoad*.

```
select avg(ev.`getDoubleValue`('load')) as YardLoad
from indenica.`events`.Event(eventType='yard.status')
   .win:length_batch(10) as ev
```

The following listing receives events about the current ratio of truck with loading job vs trucks with unloading job in the yard parking zone. The ESPER engine will collect 10 events of type *yard.status* and send an average of the *fullTracksRatio* value as a new event named *YardFullTrucksRatio*.

```
select avg(ev.`getDoubleValue`('fullTrucksRatio')) as
    YardFullTrucksRatio
from indenica.`events`.Event(eventType='yard.status')
   .win:length_batch(10) as ev
```

Appendix C

Case Study DROOLS Adaptation Rules

In this Appendix we will introduce and describe all the DROOLS adaptation rules used in the Warehouse case study.

C.1 Warehouse

In the following listings we will describe all the adaptation rules used to place some adaptation actions on the Warehouse component.
The following listing receives events of type *Usage* about the current load in the warehouse and, in case this value is equal or greater than 0.9, the Adaptation Engine will notify the Adaptation Interface in order to decrease the storage priority.

```
import indenica.events.Event;
global indenica.adaptation.component.
   AdaptationEngineImpl ae;

rule
  "Warehouse_Usage_Priority_Decrease"
when
  $event : indenica.events.Event(((Double)get('Usage'
     )) >= 0.9)
then
  Event eventOut = new Event();
  eventOut.setEventType("adaptation.warehouse.
     decreaseStoragePriority");
```

```
  ae.notifyInterface(eventOut);
end
```

The following listing receives events of type *Usage* about the current load in the warehouse and, in case this value is less or equal than 0.1, the Adaptation Engine will notify the Adaptation Interface in order to increase the storage priority.

```
import indenica.events.Event;
global indenica.adaptation.component.
   AdaptationEngineImpl ae;

rule
  "Warehouse_Usage_Priority_Increase"
when
  $event : indenica.events.Event(((Double)get('Usage'
     )) <= 0.1)
then
  Event eventOut = new Event();
  eventOut.setEventType("adaptation.warehouse.
     increaseStoragePriority");
  ae.notifyInterface(eventOut);
end
```

The following listing receives events of type *InStack* about the current incoming buffer load in the warehouse and, in case this value is greater than 0.8, the Adaptation Engine will notify the Adaptation Interface in order to decrease the conveyor speed.

```
import indenica.events.Event;
global indenica.adaptation.component.
   AdaptationEngineImpl ae;

rule
  "Conveyor_Decrease_Speed"
when
  $event : indenica.events.Event(((Double)get('
     InStack')) > 0.8)
```

```
then
  Event eventOut = new Event();
  eventOut.setEventType("adaptation.conveyor.
     decreaseSpeed");
  ae.notifyInterface(eventOut);
end
```

The following listing receives events of type *OutStack* about the current outgoing buffer load in the warehouse and, in case this value is greater than 0.8, the Adaptation Engine will notify the Adaptation Interface in order to increase the conveyor speed.

```
import indenica.events.Event;
global indenica.adaptation.component.
   AdaptationEngineImpl ae;

rule
  "Conveyor_Increase_Speed"
when
  $event : indenica.events.Event(((Double)get('
     OutStack')) > 0.8)
then
  Event eventOut = new Event();
  eventOut.setEventType("adaptation.conveyor.
     increaseSpeed");
  ae.notifyInterface(eventOut);
end
```

C.2 Yard

In the following listings we will describe all the adaptation rules used to place some adaptation actions on the Yard component.
The following listing receives events of type *OutStack* about the current outgoing buffer load in the warehouse and, in case this value is greater than 0.7, the Adaptation Engine will notify the Adaptation Interface in order to decrease the storage priority and the yard will prefer sending trucks with outgoing jobs.

```
import indenica.events.Event;
global indenica.adaptation.component.
   AdaptationEngineImpl ae;

rule
  "Yard_Decrease_Storage_Priority"
when
  $event : indenica.events.Event(((Double)get('
     OutStack')) > 0.7)
then
  Event eventOut = new Event();
  eventOut.setEventType("adaptation.yard.
     decreaseStoragePriority");
  ae.notifyInterface(eventOut);
end
```

The following listing receives events of type *InStack* about the current incoming buffer load in the warehouse and, in case this value is greater than 0.7, the Adaptation Engine will notify the Adaptation Interface in order to increase the storage priority and the yard will prefer sending trucks with incoming jobs.

```
import indenica.events.Event;
global indenica.adaptation.component.
   AdaptationEngineImpl ae;

rule
  "Yard_Increase_Storage_Priority"
when
  $event : indenica.events.Event(((Double)get('
     InStack')) > 0.7)
then
  Event eventOut = new Event();
  eventOut.setEventType("adaptation.yard.
     increaseStoragePriority");
  ae.notifyInterface(eventOut);
end
```

C.3 Loading Bay

In the following listings we will describe all the adaptation rules used to place some adaptation actions on the Loading Bay component.
The following listing receives events of type *YardLoad* about the current load in the yard parking area and of type *YardFullTrucksRatio* about the current ratio of trucks with outgoing job vs trucks with incoming job in the yard parking area. In case the *YardLoad* value is greater than 0.5 and the *YardFullTrucksRatio* value is less than 0.3, the Adaptation Engine will notify the Adaptation Interface in order to decrease the storage priority and the loading bay will prefer trucks with outgoing jobs.

```
import indenica.events.Event;
global indenica.adaptation.component.
   AdaptationEngineImpl ae;

rule
  "Yard_Loading_Priority_Out"
when
  $event : indenica.events.Event(((Double)get('
     YardLoad')) > 0.5) and $event2 : indenica.events
     .Event(((Double)get('YardFullTrucksRatio')) <
     0.3)
then
  Event eventOut = new Event();
  eventOut.setEventType("adaptation.yard.
     decreaseStoragePriority");
  ae.notifyInterface(eventOut);
end
```

The following listing receives events of type *YardLoad* about the current load in the yard parking area and of type *YardFullTrucksRatio* about the current ratio of trucks with outgoing job vs trucks with incoming job in the yard parking area. In case the *YardLoad* value is greater than 0.5 and the *YardFullTrucksRatio* value is greater than 0.7, the Adaptation Engine will notify the Adaptation Interface in order to increase the storage priority and the loading bay will prefer trucks with incoming jobs.

```
import indenica.events.Event;
global indenica.adaptation.component.
   AdaptationEngineImpl ae;

rule
  "Yard_Loading_Priority_In"
when
  $event : indenica.events.Event(((Double)get('
     YardLoad')) > 0.5) and $event2 : indenica.events
     .Event(((Double)get('YardFullTrucksRatio')) >
     0.7)
then
  Event eventOut = new Event();
  eventOut.setEventType("adaptation.yard.
     increaseStoragePriority");
  ae.notifyInterface(eventOut);
end
```