University of Crete

Computer Science Department

Monitoring QoS for Composite Web Services

Konstantina Konsolaki

Master's Thesis

Heraklion, June 2012

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ ΠΟΙΟΤΗΤΑΣ ΓΙΑ ΣΥΝΘΕΤΕΣ ΗΛΕΚΤΡΟΝΙΚΕΣ ΥΠΗΡΕΣΙΕΣ

Εργασία που υποβλήθηκε απο την:

Κωνσταντίνα Α. Κονσολάκη

ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

_____

Κωνσταντίνα Κονσολάκη, Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

_____

Δημήτρης Πλεξουσάκης, Επόπτης

_____

Αντώνης Σαββίδης, Αναπληρωτής Καθηγητής

_____

Κωνσταντίνος Μαγκούτης, Ερευνητής

Δεκτή:

_____

Άγγελος Μπίλας, Καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Ηράκλειο, Ιούνιος 2012

Monitoring QoS for Composite Web Services

Konstantina Konsolaki
Master's Thesis
Computer Science Department, University of Crete

Abstract

Web services are an emerging technology attracting a lot of attention from both academia and industry in recent years. Thus, more and more businesses adopt them to facilitate and automate their business processes. However, once services and business processes become operational, several emerging issues must be considered throughout the life-cycle of a Service-Based Application (SBA), such as the one concerning service monitoring. SBAs need to be managed and monitored, so that stakeholders have a clear view of how services perform within their operational environment and take management decisions. Many approaches have been proposed and have proven that Web service monitoring is very crucial for successful invocations.

This thesis proposes a framework for monitoring SBAs. The main component of our framework is Astro's WS-MON. Based on this component we implement a framework responsible for providing Astro with the necessary input files and also exploits the output results of this monitoring tool in order to provide the user with more monitoring properties. Furthermore, our framework checks at runtime the results of the monitors and reports the occurring violations. Finally, a specific case study is used to illustrate its functionality.

In summary, the contribution of this work lies in introducing a monitoring framework, that extends an existing tool in order to detect violations at runtime. The reported violations can be used in order for adaptation actions to take place.

Supervisor: Dimitris Plexousakis
Professor

ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ ΠΟΙΟΤΗΤΑΣ ΓΙΑ ΣΥΝΘΕΤΕΣ ΗΛΕΚΤΡΟΝΙΚΕΣ ΥΠΗΡΕΣΙΕΣ

Κωνσταντίνα Κονσολάκη
Μεταπτυχιακή Εργασία
Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Περίληψη

Οι ηλεκτρονικές υπηρεσίες είναι μια αναπτυσσόμενη τεχνολογία που τραβάει όλο και περισσότερο την προσοχή τόσο της ακαδημαϊκής κοινότητας όσο και της βιομηχανικής κοινότητας. Έτσι, όλο και περισσότερες επιχειρήσεις τις υιοθετούν προκειμένου να διευκολύνουν και να αυτοματοποιήσουν τις επιχειρησιάκες τους διαδικασίες. Παρόλα αυτά, από τη στιγμή που οι ηλεκτρονικές υπηρεσίες απέκτησαν λειτουργηκότητα, προκύπτουν κρίσιμα ζητήματα κατά τη διάρκεια του κύκλου ζωής τους, όπως αυτό που αφορά τη παρακολούθηση τους. Οι ηλεκτρονικές υπηρεσίες πρέπει να ελέγχονται και να παρακολουθούνται, έτσι ώστε τα ενδιαφερόμενα μέλη να αποκτήσουν μία σαφή όψη για το πως αποδίδουν μεσα στο λειτουργικό τους περιβάλλον και για να πάρουν αποφάσεις για τη διαχείριση τους. Πολλές προσεγγίσεις έχουν προταθεί γύρω από αυτό το θέμα και έχουν αποδείξει τη σημαντικότητα του για επιτυχημένη εκτέλεση των ηλεκτρονικών υπηρεσιών.

Η συγκεκριμένη εργασία προτείνει ένα σύστημα για τη παρακολούθηση ηλεκτρονικών υπηρεσίων. Το κύριο συστατικό του συστήματος μας είναι ο WS-MON του Astro. Με βάση αυτό το στοιχείο, υλοποιήσαμε ένα σύστημα ώστε να παρέχουμε στο Astro όλα τα απαιτούμενα αρχεία εισόδου και επίσης εκμεταλευόμαστε τα αποτελέσματα αυτού του εργαλείου παρακολούθησης προκειμένου να παρέχουμε στο χρήστη περισσότερες ιδιότητες. Επίσης το σύστημα μας, ελέγξει κατά τη διάρκεια εκτέλεσης τα αποτελέσμτα των εργαλείων παρακολούθησης και καταγράφει τις παραβιάσεις που συμβαίνουν. Τέλος, χρησιμοποιείται ένα συγκεκριμένο σενάριο ώστε να παρουσιασθεί η λειτουργικότητα του.

Συμπερασματικά, η συνεισφορά αυτής της εργασίας έγκειται στην παρουσίαση ένος συστήματος παρακολούθησης, το οποίο επεκτείνει ένα υπάρχον εργαλείο προκειμένου να καταγράφονται οι παραβιάσεις κατα τη διάρκεια εκτέλεσης. Οι παραβιάσεις αυτές είναι δυνατόν να χρησιμοποιηθούν ώστε να πραγματοποιηθούνε διορθωτικές κινήσες.

Επόπτης Καθηγητής: Δημήτρης Πλεξουσάκης

Καθηγητὴς

Ευχαριστίες

Στο σημείο αυτὸ θα ἠθελα να ευχαριστήσω τον επόπτη καθηγητή μου κ. Δημήτρη Πλεξουσάκη για την ἀψογη συνεργασία μας τα 2 τελευταία χρόνια, καθὼς επίσης και για την ουσιαστικὴ του καθοδήγηση και συμβολὴ στην ολοκλήρωση της παρούσας εργασίας.

Επίσης, θα ἠθελα να εκφράσω τις ευχαριστίες μου, στους καθηγητες κ. Αντώνη Σαββίδη και κ. Κωνσταντίνο Μαγκούτη για τη μεγάλη προθυμία τους να συμμετέχουν στην τριμελὴ επιτροπὴ.

Παράλληλα, θα ἠθελα να ευχαριστήσω τον Κυριάκο Κρητικὸ και τον Χρυσόστομο Ζεγκίνη για τη μεγάλη τους βοὴθεια και καθοδήγηση σε όλη τη διάρκεια εκπόνησης της εργασίας.

Ακόμα θα ἠθελα να ευχαριστήσω το Ινστιτούτο Πληροφορικὴς του Ιδρύματος Τεχνολογίας και Ἐρευνας για την υποστήριξη σε υλικοτεχνικὴ υποδομὴ και τεχνογνωσία.

Πολλὲς ευχαριστίες θα ἠθελα να εκφράσω στους φίλους μου, για τη στήριξη τους και για όλες τις στιγμὲς που μοιραστήκαμε μαζὶ όλο αυτὸ τον καιρὸ.

Τέλος, θα ἠθελα να ευχαριστήσω ιδιαιτέρως τους γονεὶς μου, Αντώνη και Ειρήνη και τις αδελφὲς μου, Χαρούλα και Βαγγελιώ για την υποστήριξη και την αγάπη που με περιέβαλαν.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Web services

A Web service is a software component identified by a URL, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols.

This definition has been published by the World Wide Web Consortium W3C in the Web Services Architecture document [15] and it gives a concise presentation of the advantageous characteristics of Web services, which we will briefly examine here.

The defining characteristic of a Web service is the use of the Internet and the World Wide Web as the communication medium for services to interact with each other and with service consumers. By using WWW, Web services exploit the existing URI infrastructure in order to be located by anyone having access to the Web. The URI scheme gives a name to each Web service which uniquely identifies it and allows one to use all existing operations on URIs in order to access it.

Web services make extensive use of the XML language [18]. From the definition of the messages exchanged between services to the service description, everything is based on XML, which is quite advantageous. XML is a simple language, both machine and human readable, with an intuitive hierarchical structure. It is also self-describing, as each XML data structure contains both a description of its structure and the content itself.

'A Web service can be: (i) a self-contained business task, such as funds withdrawal or funds deposit service; (ii) a full-fledged business process, such as the automated purchasing of office supplies; (iii) an application, such as a life insurance application or demand forecasts and stock replenishment; or (iv) a service-enabled resource, such as access to a particular back-end database containing patient medical records. Web services can vary in function from simple requests (e.g. credit checking and authorization, pricing enquiries, inventory status checking, or a weather report) to complete business applications that access and combine information from multiple sources, such as an insurance brokering system, an insurance liability computation, an automated travel planner, or a package tracking system [44]'.

1.1.1 Web Service Architecture

The Web service architecture consists of three entities, the service provider, the service registry and the service consumer. Fig.1.1 [1] depicts a graphical representation of the traditional Web service architecture.

The Service Provider creates or simply offers the Web service. The service provider needs to describe the Web service in a standard format, which in turn is XML and publish it in a central Service Registry. The service registry contains additional information about the service provider, such as address and contact of the providing company and technical details about the service. The Service Consumer retrieves the information from the registry and uses the service description obtained to bind to and invoke the Web service.

For realizing the above operations some technologies have to be applied. When talking about Web Services always three key-terms are mentioned, namely SOAP [17], WSDL [20] and UDDI [41]. These are XML-based technologies and represent the three core technologies of Web Services.

The basic idea of Web services is the use of SOAP messaging protocol to invoke software method in remote systems. This is often described by some technologists as Remote Procedure Calls (RPC) over the Internet protocols (e.g., HTTP). A SOAP message consists of an 'Envelope', an optional 'Header', and a mandatory 'Body'. The SOAP 'Body'

[1] https://ensweb.users.info.unicaen.fr/cours/rest/presentationM2/index.php

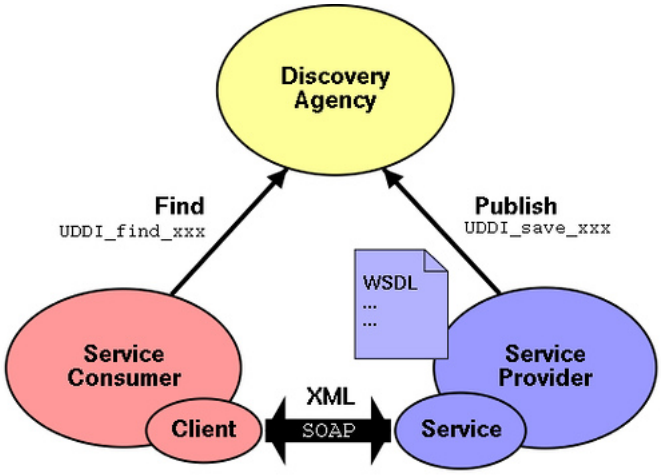


Figure 1.1: The Web Service Architecture

carries application-specific contents including the method name and the serialized values of the methods input or output parameters. Parameters of a Web services method can be a simple value or a compound value (structure or array). Serializing a Web services message in (pure text) XML format allows the SOAP XML to pass through Internet firewall.

Web services can be considered as a set of callable interfaces to software programs or components, regardless of their implementations. They can be invoked remotely via SOAP messaging. Therefore, these programs can provide services to other applications using Internet protocols.

Additional standards, WSDL and UDDI, were developed to support the description and discovery aspect of the Web services. A WSDL file contains service definitions for distributed systems to support the automatic creation of client-side stubs or proxies, and the binding to the Web services. WSDL is specified in XML format. It describes the interfaces to a Web services implementation in terms of format of the messages, binding of the abstract messages to a concrete protocol, and address of the endpoint. It is a 'take-it-or-leave-it' technical contract offered by a Web services provider to Web services consumers.

UDDI is a registry standard for Web services providers to publish their Web services. It may be used by a Web service consumer to discover (search) Web services developed by Web services providers. UDDI can store company information, services provided by a

company, and the specific technical information for binding with a specific service. The technical binding information for using a Web service will be the URL reference to the WSDL file of the Web service. The structure of UDDI repository is defined in XML Schemas containing four entity types:

- **Business Entity**, which contains information about a company;
- **Business Services**, provided by a business entity;
- **Binding Templates**, which implement business services and
- **'tModels'**, which contain references to technical specifications for services;

The Web Service Architecture is a model for building loosely coupled software services in distributed environments. The Web services network is an application level network involving a number of participants: service providers, service consumers, and service registry operators.

1.1.2 The Web Services Technology Stack

When SOAP was first developed, it was intended as a simple messaging protocol to provide remote procedure calls over the HTTP protocols. However, Web services have been gradually used to support critical business applications; therefore, additional standards, such as security and composition, have been developed or should be developed to support different aspects of Web services-based applications. Web services standards or technology stack is shown in Fig. 1.2, ([19]), to illustrate the relationships and dependencies among various Web services standards.

The bottom layer of the stack is the basic communication protocol layer for Web services including TCP/IP, HTTP, Simple Mail Transfer Protocol (SMTP), etc. XML 1.0 Specification and XML Schema are the definition languages used to define all the other Web services standards except the communication protocols. The basic messaging protocol is SOAP. There are several efforts, such as WS-ReliableMessaging, to enhance the functionality of SOAP via processing information placed in the SOAP header. The grand vision of Web services architecture is that Web services can be composed and invoked dynamically to support business processes within and across enterprises. Hence,

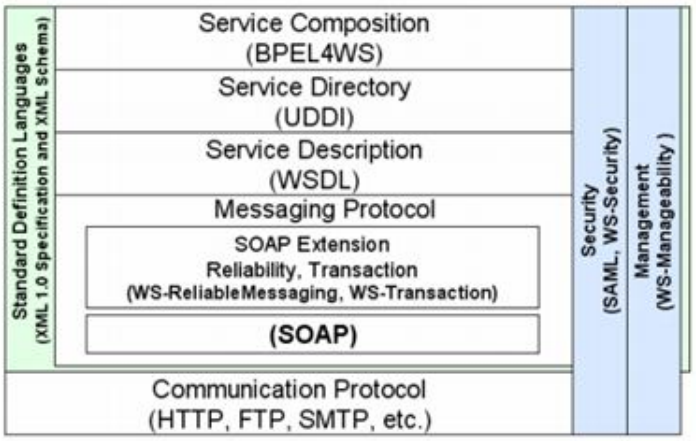


Figure 1.2: Web Services Standards Stack

the highest layer of the Web services standards stack, the Service Composition layer, has emerged. This layer consists of standards that specify how individual Web services can be composed to support business processes. A number of new languages have been introduced to address this Web services composition issue, including BPEL4WS (Business Process Execution Language for Web Services) [22], WSCI (Web Services Choreography Interface) [4], and BPML (Business Process Modeling Language) [5].

1.2 Service Based Applications

Service-Based Applications (SBA) are defined as software systems that integrate existing services, which are individually made available by different service providers, and can be accessed by service consumers, through a variety of connecting devices. SBA are open systems, as they rest on a global infrastructure i.e. the Internet on which new services may become available, or existing one disappear, and need to manage the heterogeneity of service providers and the variability of contexts and devices they can be accessed from, still maintaining their expected functionalities and qualities.

Service-based applications are often implemented in terms of an orchestration, that is, a centralized logic that describes the order in which the various services are called and the way their parameters are formed and used for further calls. This orchestration is also called Service Process.

An SBA can be illustrated by its three functional layers: i) *'Business Process Management'* (BPM), ii) *'Service Composition and Coordination'* (SCC) and iii) *'Service Infrastructure'* (SI) [30].

At the BPM layer the application activities, constraints and requirements are described without design details. The basic workflow constructed at BPM is refined at the SCC layer by the composition of suitable services. Finally, SI provides the underlying runtime environment.

Below, we will introduce the key elements of each layer.

- **BPM Layer.** Workflow and key performance indicator (KPI) are the key elements relevant for BPM. Workflow is the abstract model of the business process defining logical decision points, sequential or parallel work routes and exceptional cases. While business activities constitute the workflow, business rules together with business policies have an effect on the specification of business processes. KPI is a metric that shows quantitatively if business performance meets the pre-defined business goals.

- **SCC Layer.** Service composition, process performance metric (PPM) and service metrics are the key elements relevant for SCC. Service composition is a combination of services to realize a workflow. The designer needs to know descriptions, interfaces and supported protocols of available services for composition. PPM measures performance of a process or its parts in terms of cost, quality or duration. Service metrics are basically QoS metrics which talk about non-functional properties of services.

- **SI Layer.** Service registry, discovery and selection mechanisms constitute infrastructural facilities to find and select the services required by composition. Service registry is the information system where service descriptions are kept as a searchable repository. Service discovery and selection are the basic functionalities that serve SCC for the selection of most suitable services for SBA realization. Service realization corresponds to the run-time environment on top of where services are executed (e.g. grids, clusters, data servers, software, protocols and network infrastructure).

1.2.1 Types of services

Services exploited in a service-based application can be offered by various different agents (for instance, they can be offered by Persons or by Organizations), or they can simply be software services exploiting some specific technology, e.g., web services.

Besides for the agent that is providing them, services may also differ for their nature. They can be *abstract* when they do not have a concrete implementation but only represent an idea that could correspond, possibly in the future, to various implementations. Of course, they are *concrete* when they are actually provided by some actor. This distinction is quite relevant when developing adaptable service- based applications as a Service Integrator at design time may reason even in the absence of Concrete Services simply by exploiting Abstract Services. Clearly, in this case, the resulting application will be executable only in those cases when at runtime some Concrete Service implementing the abstract ones exists and these services are selected in some adaptation step.

Orthogonally to this classification, Services can also be distinguished in *Simple* and *Composite*. Composite Services are service-based applications being accessible as services. The current technology for building service-based applications, BPEL, actually, only supports the development of Composite Services.

The last orthogonal classification refers to the *statefulness* of services. A special kind of Stateful Services are the *Conversational Services*. These store the state of the conversation with a single specific stakeholder, but keep the states of different conversations separate from each other.

The three classifications are shown in Figure 1.3.

1.3 The need of monitoring

Software monitoring involves obtaining the information relating to the state, behavior, and environment of a software system at runtime, so as to deal with potential deviations of system behavior from requirements at the earliest possible time. Monitoring is usually carried out in parallel with the system's normal execution, without interrupting its operation. Starting from early 1960s with the advent of debuggers, software monitoring

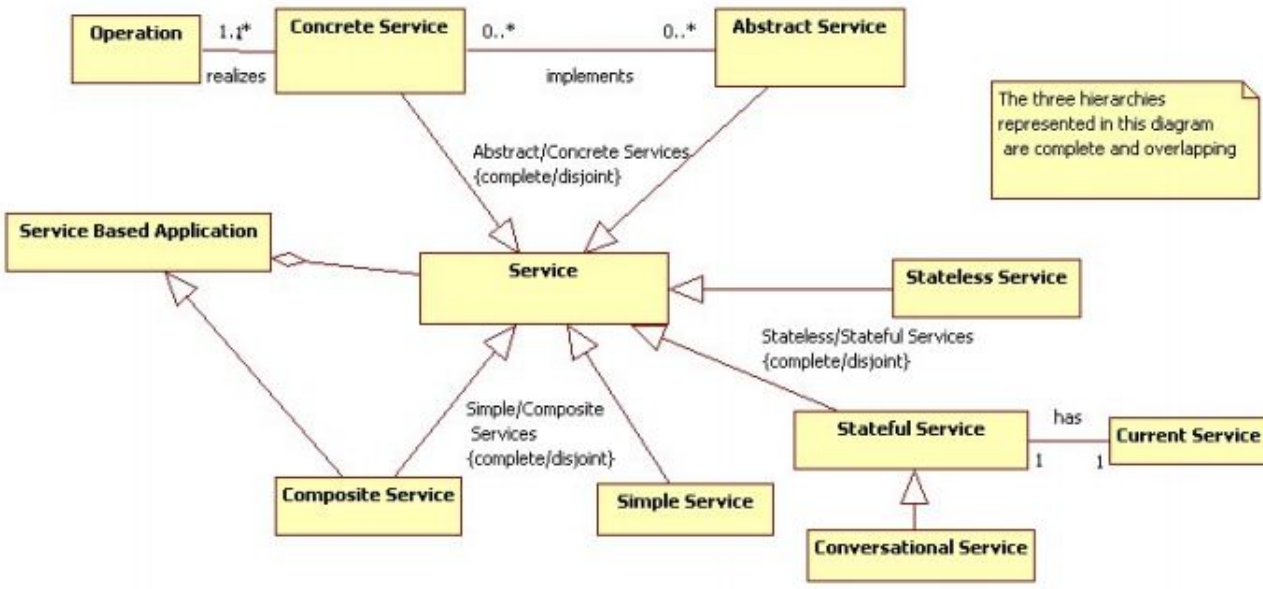


Figure 1.3: The service types diagram

has been widely used for debugging and testing, correctness checking, security and dependability analysis, performance evaluation and enhancement, and system control. A recent taxonomy shows that runtime software monitoring has been used also for profiling, software optimization, as well as software fault detection, diagnosis, and recovery.

As a special form of software, Web services also require monitoring. In particular, even if it can be demonstrated that a system can meet its requirements prior to deployment, at run-time these requirements may be violated. This may be the result of unpredicted changes in the environment of a system or failure to anticipate the behavior of all the agents interacting with it (i.e., other systems and human users).

The service monitoring phase concerns itself with service level measurement. Monitoring is the continuous and closed-loop procedure of measuring, monitoring, reporting and improving the Quality of Service (Qos) of systems and applications delivered by service-oriented solutions. Service level monitoring is a disciplined methodology for establishing acceptable levels of service that address business objectives, processes and costs.

The service monitoring phase targets continuous evaluation of service level objectives and performance. To achieve this objective service monitoring requires that a set of QoS metrics is gathered. In addition, workloads need to be monitored and the service weights

for request queues might need to be readjusted. This allows a service provider to ensure that the promised performance level is being delivered, and to take appropriate actions to rectify non-compliance with a Service Level Agreement (SLA) such as re-prioritization and reallocation of resources.

To determine whether an objective has been met QoS metrics are evaluated based on measurable data about a service - e.g., response time, throughput, availability, and so on - performance during specified times, and periodic evaluations. A key aspect of defining measurable objectives is to set warning thresholds and alerts for compliance failures. For instance, if the response time of a particular service is degrading then the step could be automatically routed to a backup service.

1.4 Quality of Service Description (QoS)

The international quality standard ISO 8402 describes quality as the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs. According to [24], what defines quality is vague, and different views exist in different studies and from different perspectives. The following three views are the most common ones:

1. *Quality as Functionality.* According to this view, quality is considered as the amount of functionality that a service can offer to its users. For example, if SP1 allows you to rent a car besides booking flights and hotel rooms, and if this functionality is not provided by SP2 or SP3, then SP1 is offering a better quality than SP2 and SP3.

2. *Quality as Conformance.* According to this view, quality means meeting specifications. For example, if SP1 has specified that his web service (WS) will be available 0.9999 of the time and this was true, then SP1 is considered as offering good quality of service.

3. *Quality as Reputation.* According to this view, quality depends on users experience and expectation from a WS and its value is built collectively over the time of the service's existence from users feedback. For example, if web services have consistently provided specific functionality with specific performance levels at all time of their operation, then they provide good quality of service.

These different views of quality require QoS to be monitored and measured differently. Quality as functionality characterizes the design of a service and can only be measured by comparing the service against other services offering similar functionalities. Quality as conformance, on the other hand, can be monitored for each service individually, and usually requires the user's experience of the service in order to measure the promise against the delivery. Finally, reputation can be regarded as a reference to a service's consistency over time in offering both functionality and conformance qualities, and can therefore be measured through the other two types of quality over time.

While it is possible to establish all three types of quality for a service in an Service Oriented Computing (SOC) environment, it is perhaps most interesting and relevant to understand how quality as conformance may be monitored and measured. The reasons for this are the following. Firstly, the first type of quality is already established in the SOC environment with the usage of the WSDL and UDDI standards, although nobody reassures that the functionality exposed by a service is the same with the one advertised by the service provider. Secondly, the reputation of a service is just an indicator of the overall QoS of the service over time that depends on user expectations and other imponderable factors (advertisement, financial and political interests, etc.). So reputation can only be considered as a QoS property that can be measured and computed by monitoring and other authorities.

Another aspect which is usually neglected is that QoS must be seen in a broader end-to-end sense as it affects the end-to-end quality received by the WS client when invoking a particular WS. Thus, QoS characterizes not only the WS but any entity used in the path from the user to the WS, including the WSs host and intervening network. Therefore, the QoS characteristics of all these entities, as they affect the end-to-end quality, constitute the QoS of a WS.

Thus, based on the above definition and aspects, we consider QoS of a WS as a set of non-functional attributes of the entities used in the path from the WS to the client that bear on the WS's ability to satisfy stated or implied needs in an end-to-end fashion. The values of QoS attributes can vary without impacting the core function of the WS which remains constant most of the time during the WS's lifetime. If a WS is advertised to have certain values (or range of values) in these QoS attributes, then we say that the WS

conforms to provide a certain QoS level.

In general, the key elements for supporting QoS in a Web services environment are summarized in what follows:

1. *Performance.* The performance of a web service represents how fast a service request can be completed. It can be measured in terms of throughput, response time, latency, execution time, and transaction time, etc.

   Throughput is the number of web service requests served in a given time interval. *Response time* is the time required to complete a web service request. *Latency* is the round-trip delay (RTD) between sending a request and receiving the response. *Execution time* is the time taken by a web service to process its sequence of activities. Finally, *transaction time* represents the time that passes while the web service is completing one complete transaction. This transaction time may depend on the definition of web service transaction.

   In general, high quality web services should provide higher throughput, faster response time, lower latency, lower execution time, and faster transaction time.

2. *Reliability.* Web services should be provided with high reliability. Reliability here represents the ability of a web service to perform its required functions under stated conditions for a specified time interval. The reliability is the overall measure of a web service to maintain its service quality. The overall measure of a web service is related to the number of failures per day, week, month, or year. Reliability is also related to the assured and ordered delivery for messages being transmitted and received by service requesters and service providers.

3. *Scalability.* Web services should be provided with high scalability. Scalability represents the capability of increasing the computing capacity of service provider's computer system and system's ability to process more users' requests, operations or transactions in a given time interval. It is also related to performance. Web services should be scalable in terms of the number operations or transactions supported.

4. *Capacity.* Web services should be provided with the required capacity. Capacity is the limit of the number of simultaneous requests which should be provided with

guaranteed performance. Web services should support the required number of simultaneous connections.

5. *Robustness.* Web services should be provided with high robustness. Robustness here represents the degree to which a web service can function correctly even in the presence of invalid, incomplete or conflicting inputs. Web services should still work even if incomplete parameters are provided to the service request invocation.

6. *Exception Handling.* Web services should be provided with the functionality of exception handling. Since it is not possible for the service designer to specify all the possible outcomes and alternatives (especially with various special cases and unanticipated possibilities), exceptions should be handled properly. Exception handling is related to how the service handles these exceptions.

7. *Accuracy.* Web services should be provided with high accuracy. Accuracy here is defined as the error rate generated by the web service. The number of errors that the service generates over a time interval should be minimized.

8. *Integrity.* Integrity for web services should be provided so that a system or component can prevent unauthorized access to, or modification of, computer programs or data. There can be two types of integrity: *data integrity* and *transactional integrity.* Data integrity defines whether the transferred data is modified in transit. Transactional integrity refers to a procedure or set of procedures, which is guaranteed to preserve database integrity in a transaction.

9. *Accessibility.* Web services should be provided with high accessibility. Accessibility here represents whether the web service is capable of serving the client's requests. High accessibility can be achieved, e.g., by building highly scalable systems.

10. *Availability.* The web service should be ready (i.e., available) for immediate consumption. This availability is the probability that the system is up and related to reliability. Time-to-Repair (TTR) is associated with availability. TTR represents the time it takes to repair the web service. The service should be available immediately when it is invoked.

11. *Conformance to standards.* Describes the compliance of a Web service with standards. Strict adherence to correct versions of standards by service providers is necessary for proper invocation of Web services by service requesters.

12. *Interoperability.* Web services should be inter-operable between the different development environments used to implement services so that developers using those services do not have to think about which programming language or operating system the services are hosted on.

13. *Security.* Web services should be provided with the required security. With the increase in the use of web services which are delivered over the public Internet, there is a growing concern about security. The web service provider may apply different approaches and levels of providing security policy depending on the service requester. Security for web services means providing authentication, authorization, confidentiality, traceability, data encryption, and non-repudiation. Each of these aspects is described below:

    - *Authentication*: Users (or other services) who can access service and data should be authenticated.
    - *Authorization*: Users (or other services) should be authorized so that they only can access the protected services.
    - *Confidentiality*: Data should be treated properly so that only authorized users (or other services) can access or modify the data.
    - *Accountability*: The supplier can be hold accountable for their services.
    - *Traceability*: It should be possible to trace the history of a service when a request was serviced.
    - *Data encryption*: Data should be encrypted.
    - *Non-Repudiation*: A user cannot deny requesting a service or data after the fact. The service provider needs to ensure this security requirement.

1.5 What a Service Level Agreement (SLA) Is

An SLA sets the expectations between the consumer and provider. It helps define the relationship between the two parties. It is the cornerstone of how the service provider sets and maintains commitments to the service consumer. A good SLA addresses five key aspects:

- What the provider is promising.
- How the provider will deliver on those promises.
- Who will measure delivery, and how.
- What happens if the provider fails to deliver as promised.
- How the SLA will change over time.

While defining an SLA, realistic and measurable commitments are important. Performing as promised is important, but swift and well communicated resolution of issues is even more important.

An SLA may have the following components:

- **Purpose**. Describing the reasons behind the creation of the SLA
- **Parties**. Describes the parties involved in the SLA and their respective roles (provider and consumer).
- **Validity period**. Defines the period of time that the SLA will cover. This is delimited by start time and end time of the term.
- **Scope**. Defines the services covered in the agreement.
- **Restrictions**. Defines the necessary steps to be taken in order for the requested service levels to be provided.
- **Service-level objective**. The levels of service that both the users and the service providers agree on, and usually include a set of service level indicators, like availability, performance and reliability. Each aspect of the service level, such as availability, will have a target level to achieve.

- **Penalties**. This field defines what sanctions should apply in case the service provider underperforms and is unable to meet the objectives specified is the SLA

- **Optional Services**. Provides for any services that are not normally required by the user, but might be required as an exception.

- **Exclusions**. Specifies what is not covered in the SLA.

- **Administration**. Describes the processes created in the SLA to meet and measure its objectives and defines organizational responsibility for overseeing each of those processes.

In a typical scenario, each web service interacts with many other web services, switching between roles of being a provider in some interactions and a consumer in others. Each of these interactions could potentially be governed by an SLA. Considering the legal and monetary implications in violating SLAs, providers need to design their SLAs only after understanding their capabilities. It is important to design SLAs that are able to balance between risk and benefit of all parties. This balance should be based on a good understanding of impact of various service levels on business processes in both the service provider and the customer.

1.6 Web Service Composition

In the emerging Web Services world companies are proceeding to offer more and more functionalities of their services as standardized Web Services, especially in the business-to-business (B2B) area. Consider the following example scenario: A company enables business partners to trace the processing of business orders online by invoking Web Services. Service consuming partners on the other hand may want to combine this service with other services they need in order to accomplish a certain goal. This aggregation of various services into a larger course of action is referred to as composition. This resulting composition may then again be offered as a new, now composite, Web Service. Two important aspects of composition are orchestration and choreography.

An orchestration defines the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function, that is, an orchestration is the

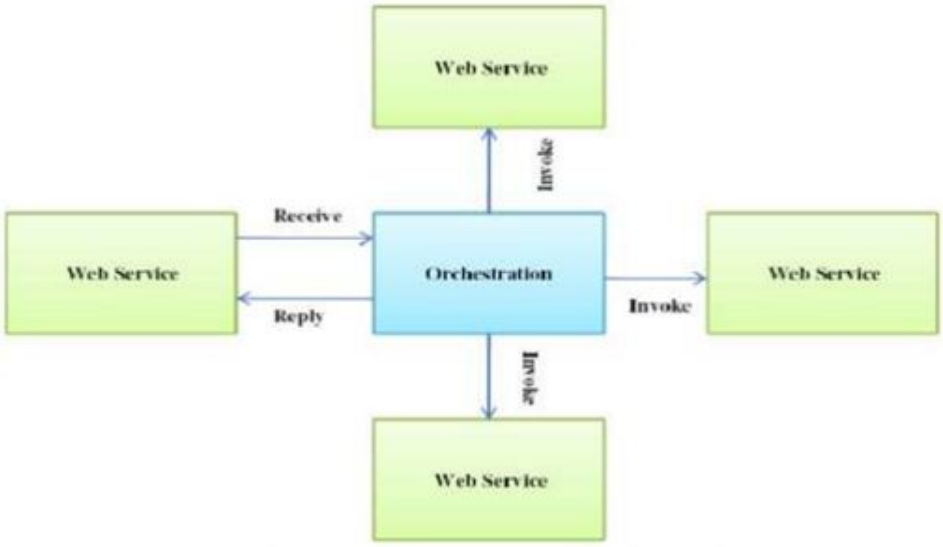


Figure 1.4: Orchestration: Web Services composed through a central process

pattern of interactions that a Web service agent must follow in order to achieve its goal. It describes how web services can interact with each other at the message level, including the business logic and execution order of the interactions. These interactions may span applications and/or organizations, and result in a long lived, transactional, multi-step process model. Furthermore, orchestration refers to an executable business process that may interact with both internal and external web services. It exposes the internal logic of a single component by specifying the control flow and data flow dependencies of that component. Fig. 1.4, ([1]), shows a number of web services that composed through a central process, the coordinator of the orchestration.

The Business Process Execution Language for Web Services (BPEL4WS) [2], which is also referred to as BPEL, is currently a de facto standard for building, specifying and executing business processes for web services composition and orchestration. BPEL composes web services to get a specific result. The composition result is named a *process*, involved services are called *partners*, and message exchange is referred to as an *activity*. In other words, a process contains a set of activities and it invokes external partner services using a WSDL interface.A BPEL process defines the order, in which involved Web services are composed, either in sequence or in parallel. BPEL allows describing conditional activities. An invocation of a Web service can for example rely on the result of another web service's invocation. With BPEL, it is possible to create loops, declare

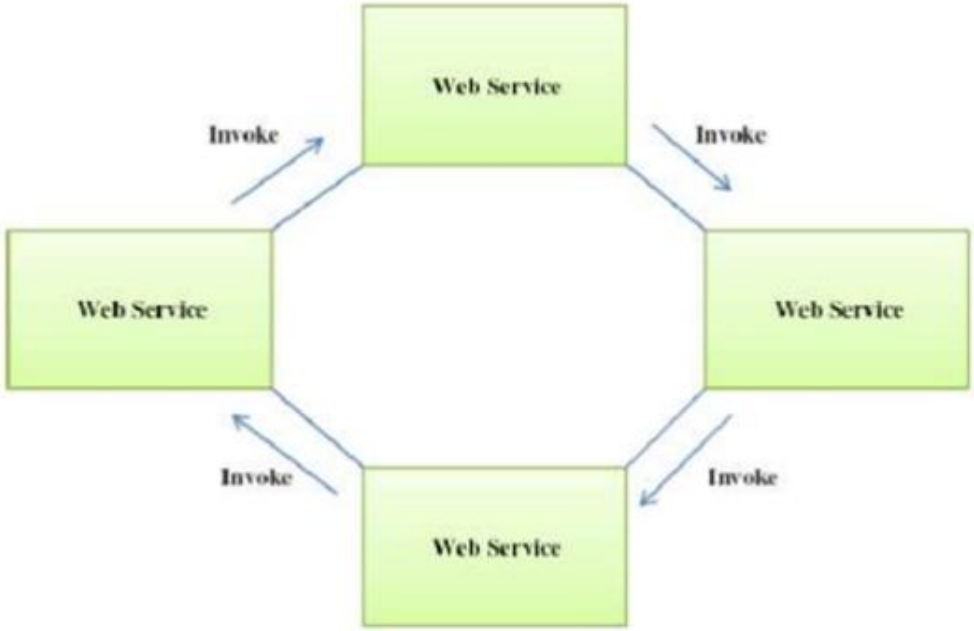


Figure 1.5: Choreography: Collaboration between services

variables, copy and assign values as well as to use fault handlers. Complex business processes can be built algorithmically by using all these constructs.

On the other hand according to the W3C's Web Services Choreography Working Group choreography is determined as the definition of the sequences and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state. Web services choreography concerns the interactions of services with their users. Any user of a Web service, automated or otherwise, is a client of that service. These users may, in turn, be other Web Services or applications. Choreography tracks the sequence of messages that may involve multiple parties and multiple sources. It is typically associated with the public message exchanges that occur between multiple web services, rather than a specific business process that is executed by a single party. Furthermore, choreography addresses the interactions that implement the collaboration between services. Multiple agents are present where each of them describes its own part in interaction. Another important thing is that service choreographies are not immediately executed but they are enacted when its participants execute their roles. Fig. 1.5, ([1]), shows the collaboration of the Web Services with choreography.

WS-CDL, the Web Services Choreography Description Language [32], is a language for describing the choreography of Web Services. WS-CDL is specified purely as a descriptive language and is not aimed at providing input for any kind of execution engine. The authors of WS-CDL use the phrase common collaborative behavior to characterize the content of a WS-CDL choreography description. A WS-CDL specification can be seen as a contract between partners which collaborate to achieve a common business goal. To overcome weaknesses of a formulation in a natural language (e.g. ambiguity), WS-CDL offers constructs for a precise description of partners and their interactions, as seen from a global viewpoint. Each participant is then responsible for conforming to this contract by developing or adjusting its services accordingly. WS-CDL promises improvement in interoperability, easy determination of conformance, and distinctness in the definition of collaborations.

In addition, Business Process Modeling Notation (BPMN) [42], is another choreography language. It is a modeling language, that supports modeling of complex control flow scenarios and also supports modeling of both private (internal) processes and abstract (public) processes. In order to model business entities and roles, BPMN uses 'pools'. It further allows combining the subsets of activities of a business process into 'swim lanes' and so makes the mapping of activities to organizational units possible. Besides allowing to describe the control flow of a participant process, BPMN aims to support modeling of the collaboration of several processes and thus choreographies. While the control flow describes behavior of each participant and specifies the ordering of the interaction-relevant activities inside of each participant (BPMN pool), the message flow 'ties' the participating processes into conversations.

BPMN and BPEL are explained in detail at section 3, since they are used for our implementation.

1.7 Contributions

In a nutshell, the main contributions of this thesis are:

- We use a tested framework for monitoring web services.
- We extend the monitoring framework so as to monitor more monitoring properties.

- We provide tools to create the files needed as input for the monitoring tool.
- We report violations at run time.
- We provide a case study in order to test our framework.
- Interesting experimental results are reported.

1.8 Organization of the thesis

Chapter 2 introduces the fundamentals about monitoring. We group the elements of the monitoring taxonomy in a way to answer the following four important questions: 'Why to monitor?', 'Who monitors?', 'What to monitor?', and 'How to monitor?'.

Chapter 3 provides a brief presentation of two web service composition languages , BPMN and BPEL, since they are used for our implementation.

Chapter 4 introduces our monitoring framework and describes in detail the implementation procedure of our work and the tools, used for it.

Chapter 5 describes the case study used in order to test our monitoring framework.

Chapter 6 presents an experimental analysis of our work. We use some simple experiments that validate our work.

Chapter 7 examines the state of the art approaches and tools used for monitoring Web services.

Chapter 8 summarizes the results of this thesis and identifies topics that are worth further work and research.

Chapter 2

Monitoring of Web services

2.1 Introduction

The term 'monitoring' has been widely used in many disciplines and in particular in service-oriented design and engineering. Depending on a particular purpose of the designed system, on the role the monitoring process plays in the system life-cycle, and the kind of information being collected, the definition of the monitoring problem has different interpretations. In a broad sense, monitoring may be defined as a process of collecting and reporting relevant information about the execution and evolution of service-based applications. This general definition becomes more concrete and clear when the monitoring goals are considered. Monitoring may be used to discover problems in the application execution. In this case monitoring may be defined as a problem of observing the behavior of a system and determining if it is consistent with a given specification [23].

Another relevant issue for the description of monitoring is what kind of data is being observed and reported, that is, the monitored information. In these regards one can consider monitoring of functional or non-functional properties, instance or class-based events, external or internal aspects, etc.

Monitoring approaches used to address one or another monitoring problem, are implemented with the help of various monitoring techniques. These approaches and techniques vary on the basis of such aspects as the way the monitoring information is specified, degree of integration with the application, timing of collecting information and information sources.

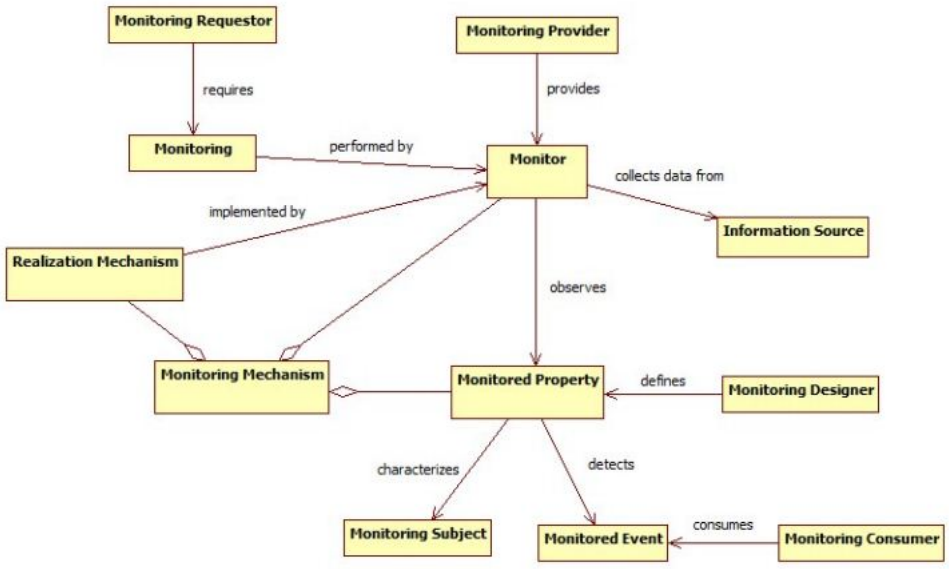


Figure 2.1: High-Level Monitoring Model

A high-level conceptual model of the monitoring concepts is represented in Fig. 2.1. As it follows from the diagram, monitoring is performed with the help of *'Monitoring Mechanisms'*, and in particular by the *'Monitors'*, which are implemented by a variety of specific *'Realization Mechanisms'*(tools and techniques). *'Monitoring Mechanisms'* include also *'Monitoring Properties'*, which allow one to identify and focus only on the important events and information. In order to observe those properties, *'Monitors'* continuously collect data from various *'Information Sources'* and detect *'Monitored Events'* corresponding to these properties. Note that this model of monitoring may have recursive implementation, in a sense that one monitor may serve as a source of information for other monitors. Depending on the purpose and the problem in hand the monitors may range from rather basic components that observe very simple properties, to rather complex monitoring frameworks capable of observing very complex properties defined with high-level specification languages.

'Monitoring properties' are used to characterize the *'Monitoring Subject"* under consideration. Depending on the approach, the *'Monitoring Subject"* may refer to the SBA itself or to its environment, to its particular elements or particular aspects of the functionality, to a particular run or to the histories of executions.

The monitoring process involves various *'Monitoring Actors'* that characterize different

roles, with which the users are involved in the process. One can identify the following types of actors:

- **Requester**, characterizes the stakeholders, who define the requirements to the system, or more precisely, to the monitoring subject.
- **Designer**, is responsible for defining the monitoring properties corresponding to the requirements of the requesters, and, if necessary, to design the corresponding monitoring approaches.
- **Provider**, represents a role in the ecosystem that owns or provides the monitoring functionalities.
- **Consumer**, is interested in results of monitoring, i.e., aims to discover important monitoring events and react to them triggering requirements for adaptation.

2.2 Monitoring Taxonomy

Relevant monitoring concepts are classified accordingly. The taxonomy aims to provide a classification for and refine the key elements of the conceptual model of SBA monitoring. We will group the elements of the monitoring taxonomy in a way to answer the following four important questions: *'Why monitor?'*, *'Who monitors?'*, *'What to monitor?'*, and *'How to monitor?'*.

2.2.1 Taxonomy Dimension: *Why?*

The 'Why?' dimension (Fig. 2.2) provides a description of the motivation for the monitoring. More precisely, the monitoring may be characterized by a particular **Usage** of the monitored information. In general sense,monitoring is used to reveal critical changes in the application or its environment, which require its adaptation. This generic purpose may have different forms depending on a particular application, domain or requirements. In particular, the following purposes of the SBA monitoring may be identified:

- *Run-time Correctness Analysis*, to check whether the execution of SBA is correct with respect certain expected specification. This may further include *Fault Monitoring*, which is used to identify different application failures, and *SLA Compliance*

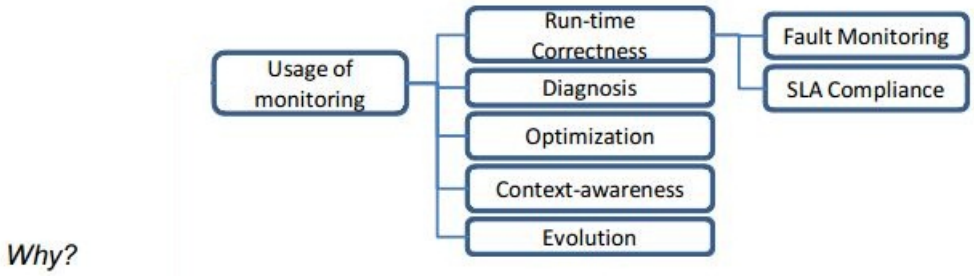


Figure 2.2: Representation of the 'why?' taxonomy.

necessary to check whether the parameters of run-time execution correspond to the service-level agreement.

- *Diagnosis*, where monitoring is used to reveal and even predict various faults in the application behavior.

- *Optimization problem*, where monitoring is used to identify a possibility for a system to work more efficiently. For this purpose, different characteristics of the SBA performance are continuously monitored.

- *Context-awareness*, where the monitored information reflects the changes in the application environment and provides necessary drivers in order to accommodate to those changes.

- *Evolution*, where the monitoring aims to observe the histories of application execution and changes in order to devise better SBA model, better adaptation mechanisms and strategies.

2.2.2 Taxonomy Dimension: *Who?*

The 'Who?' dimension (Fig. 2.3) characterizes the monitoring problem from the following points of view. First, we can characterize it from the point of view of the roles, or Actors, involved into monitoring process. We remark here also that the same physical entity may have different logical roles. Indeed, the monitoring results may be consumed by the same stakeholder who defines the monitoring requirements. Second, the monitoring may be seen from different **Perspectives**. One can distinguish:

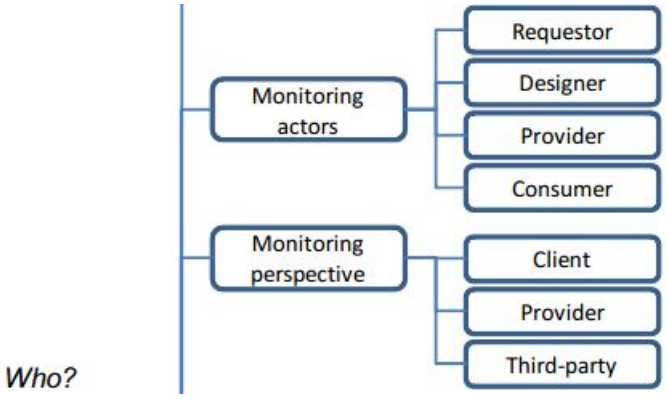


Figure 2.3: Representation of the 'who?' taxonomy.

- *client perspective*, sees the system from 'outside', aiming to check whether it delivers what is expected by the customers.
- *provider perspective*, helps to understand whether it is appropriate 'inside', i.e., satisfy the expectations of the system owner.
- *third-party perspective*, takes an independent view on the subject of monitoring.

Note that these two aspects are rather orthogonal: the monitoring requirements may come from either client- or provider-site; the monitoring mechanisms may be provided together with the system (provider perspective), installed by the consumers (client perspective), or provided by third parties.

2.2.3 Taxonomy Dimension: *What?*

The 'What?' dimension (Fig. 2.4) is used to classify the subject of monitoring and the way it is described. In this way, we consider the following elements of the taxonomy: *Monitoring Subject*, *Monitoring Aspect*, and *Monitored Property*. For the **Monitoring Subject** at the highest level we distinguish:

- *SBA Instance*, corresponding for instance to a particular BPEL process run, an application customized to a particular user according to her user profile, a particular configuration of a service composition, etc;
- *SBA Class* that define the whole application model, including its business process model, business requirements and KPIs;

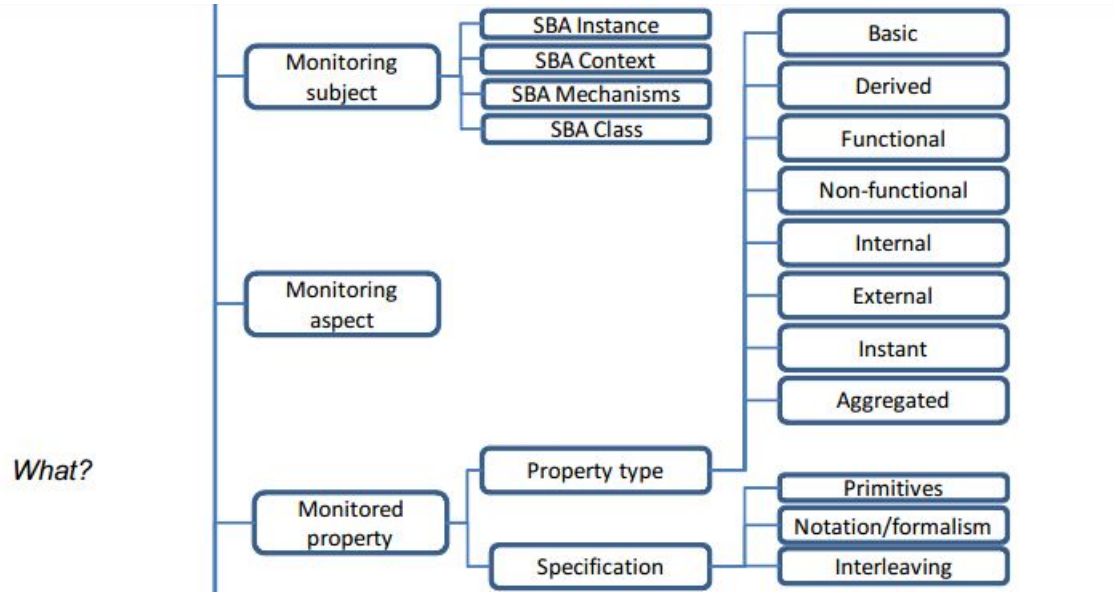


Figure 2.4: Representation of the 'what?' taxonomy.

- *SBA Context*, that describe the operational and information environment of the application;
- *Adaptation and Monitoring Mechanisms*, providing a feedback over the way the system is observed, changed, and managed.

These elements may be further decomposed into the elements with finer granularity, e.g., services, compositions, infrastructural elements, traces, locations, etc.

Monitoring Aspect refers to a particular concern of the monitored system relevant for the monitoring requester. Such aspects may refer, to different dimensions of the SBA quality model (e.g., security, dependability, usability), to the functional correctness of the system, to Service-Level Agreements, user- related information and HCI aspects, business-level metrics, KPIs, and requirements.

Monitored Property provides a way to represent these aspects of the monitored system. We further classify monitored properties according to the type of the properties and to their specification. **Property Types** define various characteristics of monitored properties. We distinguish:

- basic or derived;
- functional or non-functional properties;

- internal or external properties;

- instant or aggregated properties.

Basic properties refer to the elementary primitives and events, while *derived* properties are recursively defined on top of other properties. *Functional* properties characterize the function (or behavior) that a given system is expected to provide. Typical examples of the functional properties are failures, assertions or behavioral properties, invariants. *Non-functional* properties define quality characteristics that often can be measured in a quantitative way. Typical non-functional properties refer to availability, latency, reliability. *Internal* properties refer to the characteristics internal to the application. On the contrary, *external* properties describe the environment of the application or its context, whatever notion of the context is exploited. *Instant* properties refers to the observations performed in a particular moment of time, while the *aggregated* properties characterize the whole execution, sets of executions or event evolution of the system collecting and aggregating historical data.

Specification of Monitored Properties characterizes the languages means used to define the properties of interest. The relevant elements of such a classification are:

- *monitoring primitives*, i.e., basic building blocks used to define more complex derived properties. A typical example is the event property, which refers to elementary events mentioned in the monitored specification.

- *notation and formalism* used to unambiguously express the required properties.

- *level of abstraction* from the implementation and domain-specific details.

- *degree of interleaving* with the application specification that characterize how tight the relation between the monitoring specification and application specification is. This may range from cases, where the monitoring specification is a part of application logic, to the cases, where it is defined and changed completely separately from the application logic.

2.2.4 Taxonomy Dimension: *How?*

The way the monitoring approach is delivered may be further classified according to how it is defined and is supposed to work (*Monitoring Methodology*), how it is structured (*Monitoring Architecture*), and how it is realized (*Monitoring Implementation*). Fig. 2.5 depicts the graphical representation of the 'how?' monitoring taxonomy

Monitoring Methodology defines a set of characteristics of the monitoring process itself. It describes, in particular:

- *Information gathering*, i.e., the approach used to collect and if necessary to aggregate data from various information sources. One can distinguish between polling mode of information gathering, when, e.g., the sources are periodically queried, push mode, when the information gathering is event-driven, or more sophisticated simulation mode: a certain model of the monitored property continuously evolve on the basis of the information and events collected in either of the two previous modes.

- *Timeliness*, i.e., the characteristic of the time difference between the moment, when the event actually takes place, and the moment it is reported by the monitor. In these regards, one can distinguish reactive monitoring approaches, which aim to report events as soon as it is possible, post-mortem approaches, which report information considerably after the events (or even series of events) take place, and pro-active approaches that try to predict the occurrence of events.

- *Execution*, i.e., the characteristics of the monitoring process with respect to the system execution process. One can distinguish between blocking (or synchronous) approaches, where the execution of the monitoring subject is blocked until all the monitoring measurements are done, and non-blocking (or asynchronous) approaches, where the monitoring process is performed in parallel with the execution / evolution of the monitoring subject.

- *Monitoring Techniques*, i.e., particular solutions exploited in order to provide the above characteristics of the monitoring process. Data or process mining, database monitoring, automata-theoretic approaches to define the logics of the monitor model are the examples of such techniques.

Monitoring Architecture defines the way the monitoring framework is structured and decomposed. The relevant characteristics of this architecture are:

- *Distribution*, i.e., 'horizontal' structuring of the monitoring framework. It defines how the components of the framework are logically and physically located. We distinguish between centralized architectures, where the monitoring components are concentrated in a single node, and distributed architectures, where the monitoring components are distributed across the network, according, e.g., to the distribution of SBA components.

- *Functional SBA Layers* involved in the monitoring, i.e.,'vertical' structuring of the monitoring framework. The monitoring framework may be built on top of the single components and elements provided in business process management layer, service composition layer, and service infrastructure layer, or may involve sets of those components across functional layers (cross-layer monitoring).

- *Invasiveness*, i.e., characteristic of the monitoring framework from the perspective of how tightly it is integrated with the monitoring subject. We distinguish between the cases, when the monitoring facilities are integrated with the subject, the cases, when the monitoring facilities are integrated with the platform, where the subject operates, and the cases, when the monitoring facilities are completely separated and independent from the subject of monitoring.

Monitoring Implementation defines the way the monitoring methodology and architecture are realized. It is characterized by the *Information Sources*, the *Realization Mechanisms*, and the *Monitoring Infrastructure*.

- *Information Sources* represent various components and entities that provide all the data, which is used by the monitor in order to evaluate the monitored properties. These sources may range from rather basic elements (such as messages, log files, or timers), to more complex monitors based on top of them (sensors, probes), to hierarchically complex monitoring systems, thus providing recursive and reusable monitoring solutions. In other words, one monitor may re-use another monitor as a source if information, where the information are the events reported by the latter.

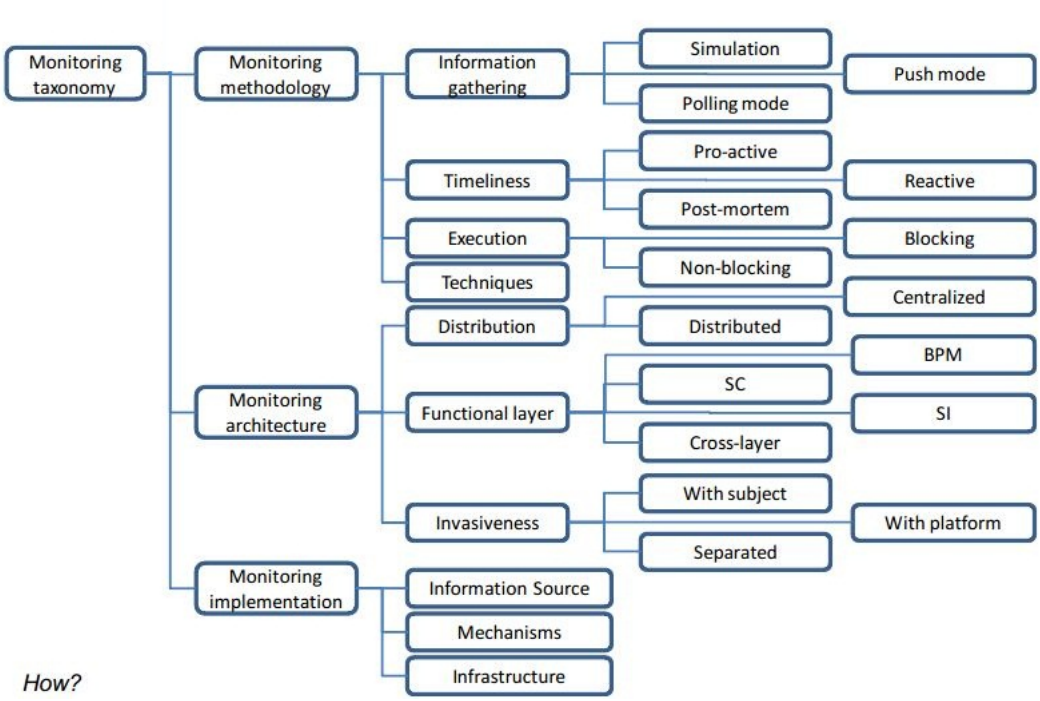


Figure 2.5: Representation of the 'how?' taxonomy.

- *Realization Mechanisms* define the tools and facilities, necessary to enable a given monitoring methodology, to implement the monitoring techniques, and to build the corresponding monitoring architecture. As it follows from this generic definition, realization mechanisms strongly depend on a given monitoring problem and on the approach used for that. Typical examples include, in particular, aspect-oriented programming techniques that enable injection of monitors into the application or to the platform code; automatic generators of monitoring programs that are used to device executable monitors from high-level monitoring specifications; dynamic monitoring solutions, which enable on-the-fly modifications of the way the monitoring of a given subject is performed, e.g., by changing the set of monitored properties or their priorities.

- *Monitoring Infrastructure* refers to the tools and facilities that provides a basis for the monitoring framework. It includes services and APIs for relating to specific information sources, for accessing and managing other monitors, containers and execution platforms to deploy and execute monitoring code, etc. As in the case of

realization mechanisms, these functionalities may be very specific for various monitoring approaches.

2.3 Monitoring Challenges

In domain of service-based applications, where the implementation and management of the underlying services is not under the control of the system integrator, important application characteristics, such as correctness, often can be evaluated only at the production settings, and require dynamic means of the analysis. This leads to an increasing importance and spread of monitoring techniques in the development and procurement of the service-oriented architectures. A wide range of research and industrial approaches towards run-time monitoring of service-based systems have been developed. These approaches target the problem from different perspectives, and provide a broad spectrum of means for observing various application aspects at different functional SBA layers.

In spite of successful development in this area, the research community still has to face a lot of very important challenges in order to be able to deliver complete and mature monitoring solutions. One of the main problems in these regards relates to high fragmentation and specificity of the proposed approaches. A typical situation is that the approach addresses a very specific aspect of the system; it targets only a particular functional layer of the application in isolation of other aspects and components. As a consequence, the approaches often come up with a very specific, ad-hoc specification languages and methodologies, as well as specific and hardly adoptable implementation solutions. This not only makes the application of many monitoring approaches difficult in practice, but gives rise to another critical problem, the one of application diagnosability. Indeed, in order to be able to identify the real source of the detected problem, it is necessary to observe the execution of the application from all its aspects and to understand and infer dependencies between these aspects.

The new approaches to the problem of monitoring service-based applications should come up with holistic and comprehensive methodologies that:

- Integrate various monitoring techniques and methods at all the functional SBA layers;

- Provide a way to target all the relevant application aspects and information;
- Define rich and well-structured modeling and specification languages capable of representing these aspects;
- Allow for modeling, identifying, and propagating dependencies and effects of monitored events and information across various functional layers and aspects in order to enable the diagnosability of applications.

Chapter 3

Web Service Composition Languages

3.1 An Introdution to BPEL4WS

In mid 2002, the first version of BPEL4WS (formerly BPEL) was developed by IBM, BEA and Microsoft. It is an XML based language that utilizes several standards like XPath [13], WS-Addressing [16] and WSDL. BPEL only allows Web services as its building blocks and does not support human interaction (BPEL was designed for pure machine to machine communication). IBM addresses these limitations by providing two standards that extend the BPEL specication: BPELJ [14] and BPEL4People [33].

The BPEL specification distinguishes two different kinds of business processes: **executable processes** and **abstract processes**. Whereas executable processes must contain all the details that are necessary to be executed by a BPEL engine, abstract processes are not executable and can have parts where things are left unspecified or explicitly marked as opaque (i. e., hidden)

3.1.1 Basic structure of a BPEL process

Fig. 3.1, ([48]), depicts the core structure of a BPEL process, and how it interacts with components external to it; either web services that the BPEL process invokes (Service A and Service B in this case) or external clients that invoke the BPEL process as a web service. The BPEL process is divided into two distinct parts; the Partner Links, which describe the interactions between the BPEL process and the outside world; and the core BPEL process itself, which describes the process to be executed at run time.

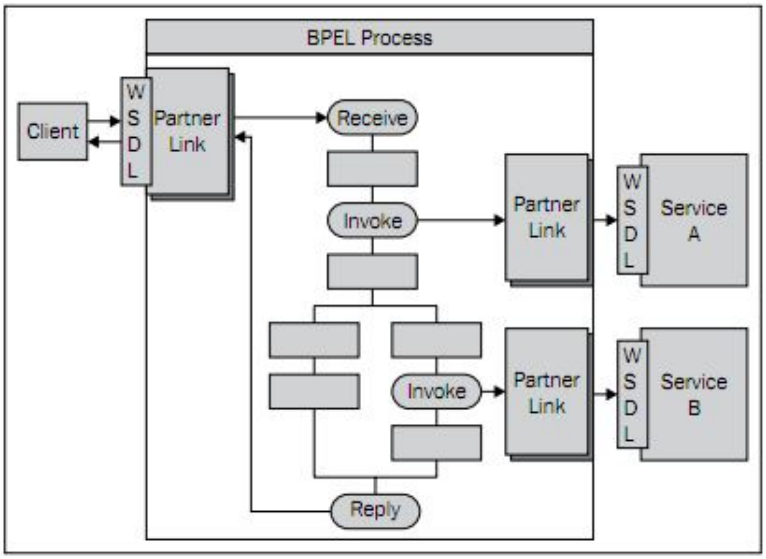


Figure 3.1: Core structure of a BPEL process

The core BPEL process consists of a number of steps, or activities as they are called in BPEL.These consist of simple activities, including:

- **Assign**: It is used to manipulate variables.
- **Transform**: It is a specialized assign activity that uses XSLT to map data from a source format to a target format.
- **Wait**: It is used to pause the process for a period of time.
- **Empty**: It does nothing. It is used in branches where syntactically an activity is required, but there is no need to perform one.

BPEL also consists of the structured activities which control the flow through the process. These include:

- **While**: It is used for implementing loops.
- **Switch**: It is a construct for implementing conditional branches.
- **Flow**: It is used for implementing branches which execute in parallel.
- **FlowN**: It is used for implementing a dynamic number of parallel branches.

The activities within a BPEL process can be sub-divided into logical groups of activities, using the **Scope** activity. As well as providing a useful way to structure and organize your process, it also enables you to define attributes such as variables, fault handlers, and compensation handlers that just apply to the scope.

In addition, each BPEL process also defines **variables**, which are used to hold the state of the process as well as messages that are sent and received by the process. They can be defined at the process level, in which case they are considered global and visible to all parts of the process. Or it can be declared within a Scope in which case they are only visible to activities contained within that Scope. Variables can be one of the following types:

- **Simple Type**: It can hold any simple data type defined by XML Schema (for example string, integer, boolean, and float).
- **WSDL Message Type**: It is used to hold the content of a WSDL Message sent to or received from partners.
- **Element**: It can hold either a complex or simple XML Schema element defined in either a WSDL file or a separate XML Schema.

Variables are manipulated using the **'assign'** activity, which can be used to copy data from one variable to another, as well as create new data using XPath Expressions or XSLT.

All interaction between a process and other parties (or partners) is via web services as defined by their corresponding WSDL files. Even though each service is fully described by its WSDL, it fails to define the relationship between the process and the partner, that is who the consumer of a service is and who the provider is. On first appearance, the relationship may seem implicit; however, this is not always the case. BPEL uses **Partner Links** to explicitly define this relationship. Partner Links are defined using the **'partnerLinkType'** which is an extension to WSDL.

Moreover, BPEL defines three messaging activities **'receive'**, **'reply'**, and **'invoke'**. The use of these activities depends on whether the message interaction is either *synchronous* or *asynchronous* and whether the BPEL process is either a consumer or provider of the service.

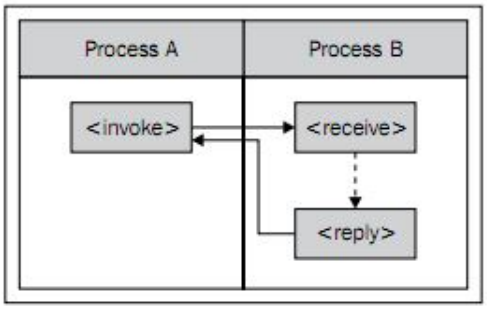


Figure 3.2: Synchronous Messaging

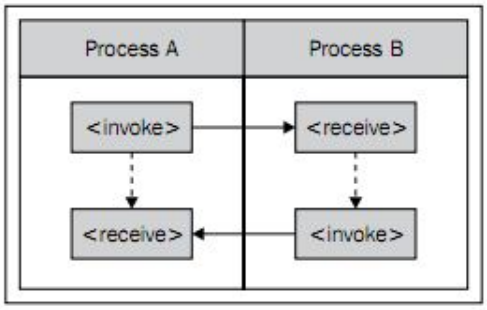


Figure 3.3: Asynchronous Messaging

With synchronous messaging, the caller will block until it has received a reply or times out. In this case the BPEL process will wait for a reply before moving onto the next activity. As presented in Fig. 3.2, ([48]), Process A uses the "invoke" activity to call a synchronous web service (Process B in this case) and once it has sent the initial request, it blocks and waits for a corresponding reply from Process B. Process B, uses the 'receive' activity to receive the request and the 'reply' activity to send a response back to Process A.

On the other hand, with asynchronous messaging, the key difference is that once the caller has sent the request, the send operation will return immediately, and the BPEL process may then continue with additional activities until it is ready to receive the reply. In Fig. 3.3, ([48]), Process A uses the 'invoke' activity to call an asynchronous web service, but contrary to synchronous request it does not block waiting for a response, but continues processing until it is ready to process the response ('receive' activity). Conversely, Process B uses a 'receive' activity to receive the initial request and an 'invoke' activity to send back the corresponding response.

3.2 An Introduction to BPMN

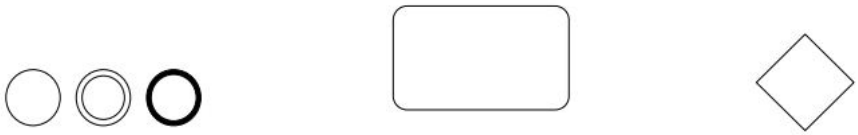
The Business Process Management Initiative (BPMI) has developed a standard Business Process Modeling Notation (BPMN). The primary goal of the BPMN effort was to provide a notation that is readily understandable by all business users, from the business analysts who create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and, finally, to the business people who will manage and monitor those processes.

BPMN defines a Business Process Diagram (**BPD**), which is based on a flowcharting technique tailored for creating graphical models of business process operations. A Business Process Model, then, is a network of graphical objects, which are activities (i.e., work) and the flow controls that define their order of performance.

3.2.1 BPMN Basics

A BPD is made up of a set of graphical elements. These elements enable the easy development of simple diagrams that will look familiar to most business analysts (e.g., a flowchart diagram). The elements were chosen to be distinguishable from each other and to utilize shapes that are familiar to most modelers. For example, activities are rectangles, and decisions are diamonds. It should be emphasized that one of the drivers for the development of BPMN is to create a simple mechanism for creating business process models, while at the same time being able to handle the complexity inherent to business processes. The approach taken to handle these two conflicting requirements was to organize the graphical aspects of the notation into specific categories. This provides a small set of notation categories so that the reader of a BPD can easily recognize the basic types of elements and understand the diagram. Within the basic categories of elements, additional variation and information can be added to support the requirements for complexity without dramatically changing the basic look-and-feel of the diagram. The four basic categories of elements are:

- Flow Objects
- Connecting Objects
- Swimlanes



(a) Types of Events. (b) Activity. (c) Gateway.

Figure 3.4: Flow Objects.

- Artifacts

3.2.1.1 Flow Objects

A BPD has a small set of core elements, which are the *Flow Objects*, so that modelers do not have to learn and recognize a large number of different shapes. The three Flow Objects are:

1. **Event**. An *Event* is represented by a circle and is something that 'happens' during the course of a business process. These Events affect the flow of the process and usually have a cause (trigger) or an impact (result). Events are circles with open centers to allow internal markers to differentiate different triggers or results. There are three types of Events, based on when they affect the flow: Start, Intermediate , and End (Fig. 3.4a respectively).

2. **Activity**. An *Activity* is represented by a rounded-corner rectangle (Fig. 3.4b) and is a generic term for work that company performs. An Activity can be atomic or non-atomic (compound). The types of Activities are: Task and Sub-Process. The Sub-Process is distinguished by a small plus sign in the bottom center of the shape.

3. **Gateway** A *Gateway* is represented by the familiar diamond shape (Fig. 3.4c) and is used to control the divergence and convergence of Sequence Flow. Thus, it will determine traditional decisions, as well as the forking, merging, and joining of paths. Internal Markers will indicate the type of behavior control.

3.2.1.2 Connecting Objects

The Flow Objects are connected together in a diagram to create the basic skeletal structure of a business process. There are three *Connecting Objects* that provide this

(a) Sequence Flow. (b) Message Flow. (c) Association.

Figure 3.5: Connecting Objects.

(a) Pool. (b) Lane.

Figure 3.6: Swimlanes.

function.These connectors are:

1. **Sequence Flow** A *Sequence Flow* is represented by a solid line with a solid arrowhead (Fig. 3.5a) and is used to show the order (the sequence) that activities will be performed in a Process. Note that the term control flow is generally not used in BPMN.

2. **Message Flow** A *Message Flow* is represented by a dashed line with an open arrowhead (Fig. 3.5b) and is used to show the flow of messages between two separate Process Participants (business entities or business roles) that send and receive them. In BPMN, two separate Pool in the Diagram will represent the two Participants.

3. **Association** An *Association* is represented by a dotted line with a line arrowhead (Fig. 3.5c) and is used to associate data, text, and other Artifacts with flow objects. Associations are used to show the inputs and outputs of activities.

3.2.1.3 Swimlanes

Many process modeling methodologies utilize the concept of swimlanes as a mechanism to organize activities into separate visual categories in order to illustrate different functional capabilities or responsibilities. BPMN supports *swimlanes* with two main constructs. The two types of BPD swimlane objects are:

1. **Pool**. A *Pool* represents a Participant in a Process. It is also acts as a graphical container for partitioning a set of activities from other Pools (Fig. 3.6a).

(a) Data Object.

(b) Group.

(c) Annotation.

Figure 3.7: Artifacts.

2. **Lane**. A *Lane* is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally (Fig. 3.6b). Lanes are used to organize and categorize activities.

Pools are used when the diagram involves two separate business entities or participants and are physically separated in the diagram. The activities within separate Pools are considered self-contained Processes. Thus, the Sequence Flow may not cross the boundary of a Pool. Message Flow is defined as being the mechanism to show the communication between two participants, and, thus, must connect between two Pools (or the objects within the Pools).

Lanes are more closely related to the traditional swimlane process modeling methodologies. Lanes are often used to separate the activities associated with a specific company function or role. Sequence Flow may cross the boundaries of Lanes within a Pool, but Message Flow may not be used between Flow Objects in Lanes of the same Pool.

3.2.1.4 Artifacts

BPMN was designed to allow modelers and modeling tools some flexibility in extending the basic notation and in providing the ability to add context appropriate to a specific modeling situation, such as for a vertical market (e.g., insurance or banking). Any number of *Artifacts* can be added to a diagram, as appropriate for the context of the business processes being modeled. The BPMN specification predefines only three types of BPD Artifacts, which are:

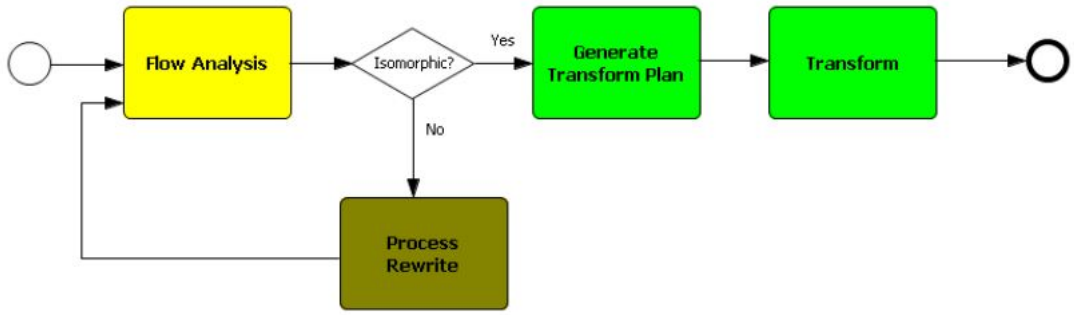


Figure 3.8: BPMN Transformation Workflow

1. **Data Object**. *Data Objects* (Fig.3.7a) are a mechanism to show how data is required or produced by activities. They are connected to activities through Associations.

2. **Group**. A *Group* is represented by a rounded corner rectangle drawn with a dashed line (Fig. 3.7b). The grouping can be used for documentation or analysis purposes, but does not affect the Sequence Flow.

3. **Annotation**. *Annotations* are a mechanism for a modeler to provide additional text information for the reader of a BPMN Diagram (Fig. 3.7c).

3.3 Mapping a BPMN Diagram to a BPEL4WS

BPMN diagrams can be mapped to Business Process Execution Language (BPEL) processes to bridge the gap between business process design and implementation. However, it is intrinsically complex to map the diagrams to BPEL processes because of the structural disparity between BPMN and BPEL. BPEL is a block structured language overall, even though a flow with links in BPEL can be more flexible. In contrast, BPMN is a constrained, but relative free form graph. Structurally, BPMN can be a super- set of BPEL. There are no fundamental difficulties in mapping a BPEL process to an isomorphic BPMN diagram. In other words, any BPEL process can be visualized as a BPMN diagram without rearranging the flows. But it is not always possible to map a BPMN diagram directly to an isomorphic BPEL process. Arbitrary sequence flows allowed in BPMN are similar to the GOTO statements in some computer languages. Without analyzing and redrawing

such diagram flow structures, it is practically impossible to map all processes correctly.

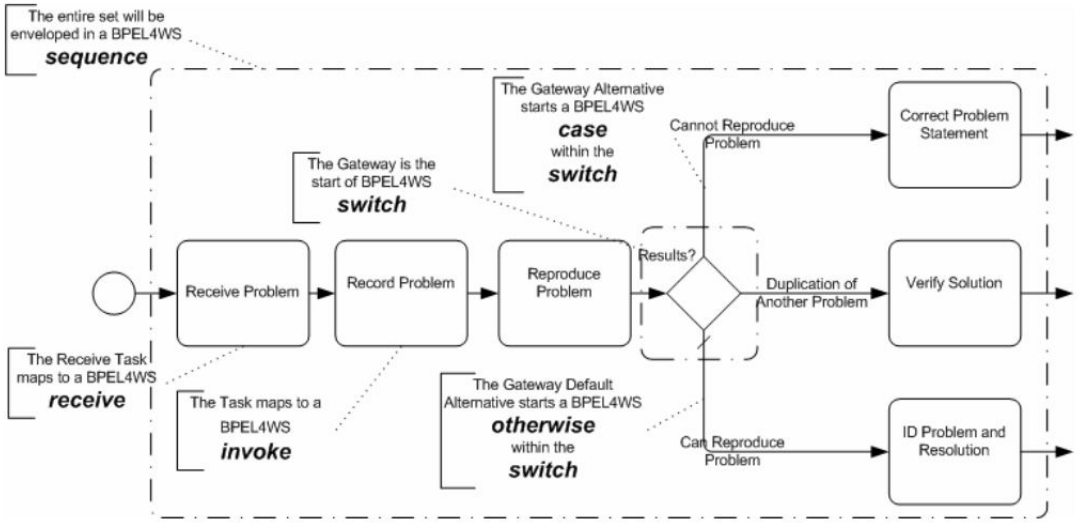


Figure 3.9: A BPD with Annotations to Show the Mapping to BPEL4WS

The process of validating, redrawing and transforming BPMN is illustrated as a workflow in Fig.3.8, ([29]). Note it is critical to analyze if a BPMN diagram can be isomorphically mapped to BPEL. And it is technically challenging to rewrite arbitrary BPMN diagrams to be BPEL isomorphic.

Finally, Fig. 3.9, ([47]), provides an example of a segment of a business process and marks the mapping to the BPEL4WS execution elements.

3.3.1 Industrial Tools for Mapping BPMN to BPEL

A more detailed mapping of BPMN to BPEL has been implemented in a number of tools.

ADONIS:Community Edition

ADONIS:Community Edition is a free version of its BPM-tool ADONIS. It is aimed both at users new to Business Process Management, as well as experienced BPM practitioners. It supports BPMN and features BPEL and XML exports for easy integration with other tools and systems. The ADONIS Community Portal [1] offers the free download of the ADONIS:Community Edition.

[1] www.adonis-community.com

Intalio

Intalio [2] provides an interesting alternative for business process modeling (BPM) tools. It provides official support for BPEL and BPM projects. Intalio Designer is the tool for modeling a business process with BPMN and this notation can be transformed to BPEL by the Designer.

BPMN2BPEL

BPMN2BPEL [3] is an open source Eclipse plugin. The translator is implemented according to [43].

Enterprise Architect

BPEL is supported in the Business and Software Engineering and Ultimate editions of Enterprise Architect [4]. It currently supports generating BPEL from executable processes. With the help of the BPMN version 1.1 Profile, Enterprise Architect enables you to develop BPEL diagrams quickly and simply.

Babel BPMN2BPEL

Babel BPMN2BPEL [5] is part of the process transformation tools developed in the BABEL project (project developed by the Business Process Management group at QUT). Babel BPMN2BPEL is the tool used in our implementation and is further explained at section 4.

[2] http://www.intalio.com
[3] http://code.google.com/p/bpmn2bpel/
[4] http://www.sparxsystems.com
[5] http://www.bpm.scitech.qut.edu.au/research/projects/oldprojects/babel/tools/

Chapter 4

Monitoring Framework

In this chapter, we introduce the monitoring framework of our implementation, presented in Fig. 4.1. The main component of our framework is Astro's WS-MON [7]. Based on this component we implement a framework responsible for providing Astro with the necessary input files and also exploits the output results of this monitoring tool in order to provide the user with more monitoring properties. The input files needed for Astro are: the **abstract BPEL** processes, the corresponding **WSDL documents** and also a **choreography** (.chor) file, which contains the composition's partners and also the definition of the monitoring properties. Furthermore, our framework checks at runtime the results of the monitors and reports the occurred violations.

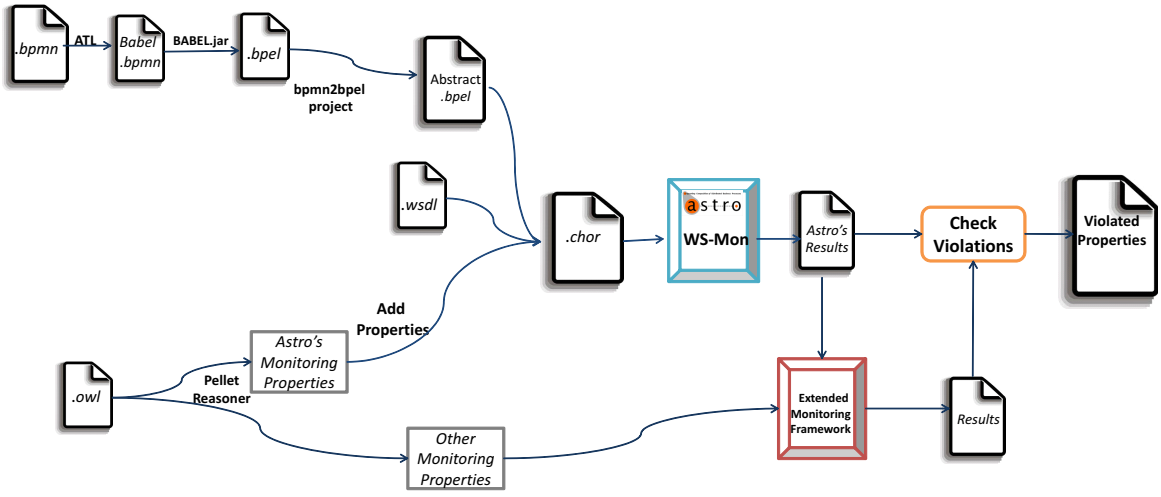


Figure 4.1: Our Monitoring Framework.

The next subsections present in detail the various parts of our monitoring framework.Fig.4.2

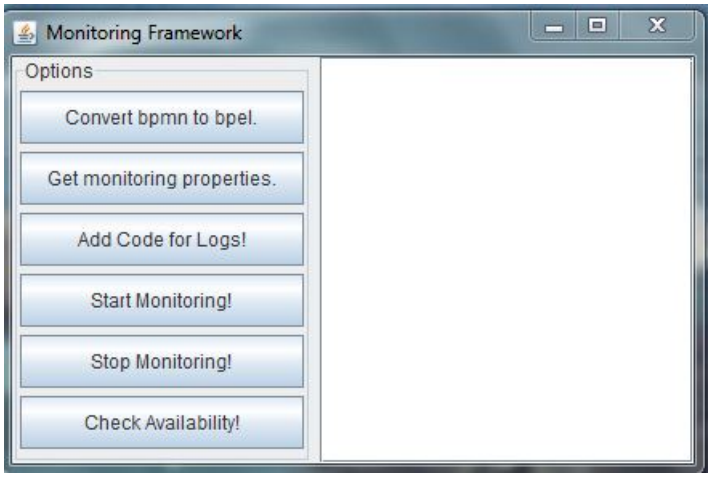


Figure 4.2: Main Window of Monitoring Framework.

depicts the options our monitoring framework provides and will also be analyzed below.

4.1 Transform BPMN to Abstract BPEL Process

For the purpose of our implementation, we choose to provide the user the ability to design BPMN models rather than abstract BPEL processes. BPMN enables the business user to develop readily understandable graphical representations of business processes and also is supported with appropriate graphical object properties that will enable the generation of BPEL processes. BPMN is a significant simplification over existing notations used for BPEL. The BPMN model is then mapped to the corresponding abstract BPEL process, used as input to the Astro's monitoring tool.

This step of our implementation requires several technologies that work together:

- **Eclipse**[1]**.** Eclipse is the master platform upon which the Eclipse SOA Tools Platform Project (STP) plug-in runs on.

- **Eclipse SOA Tools Platform (STP) plugin** [2]. The mission of STP project is to build frameworks and exemplary extensible tools that enable the design, configuration, deployment and management of software designed around a Service Oriented Architecture It includes the BPMN subproject which provides an editor and a set of tools to model business processes diagrams using the BPMN notation.

[1] **http://www.eclipse.org**
[2] **http://www.eclipse.org/stp/**

- **Eclipse ATLAS Transformation Language (ATL)** [3] ATL is a model transformation language and toolkit. In the field of Model-Driven Engineering (MDE), ATL provides ways to produce a set of target models from a set of source models. In our implementation, we will use ATL in order to perform a transformation, converting Eclipse STP's BPMN to BABEL BPMN.

- **BABEL Tools** [4] The BPMN2BPEL is a tool for translating process models represented in BPMN into process definitions represented in BPEL.

4.1.1 Implementation Procedure

In this section, we analyze the procedure needed to generate the BPEL process. It consists of three steps that are given in detail below:

1. *Creation of the BPMN diagram using the Eclipse BPMN Modeler.*

2. *Transformation of Eclipse STP's BPMN to BABEL's BPMN.* Create a new project within the Eclipse ATL to house the transformation rules. In this step, three more files are needed:

   - *bpmn.ecore.* An EMF file defining the structure for BABEL BPMN files.
   - *stpmodel.ecore.* An EMF file defining the structure for STP BPMN files.
   - *bpmn2babel.atl.* An ATL file defining the transformation from STP BPMN to BABEL BPMN.

   To proceed, click Run > Open Run Dialog. In the left-hand column, right-click ATL Transformation, and click New. Name the launch configuration (Fig. 4.3) and in the project pane select your project and the bpmn2babel.atl file Then select the BPMN and BABEL Ecore metamodels in the metamodels panel. Moreover, choose the source model and since the target model doesn't exist yet copy the source model file to the target model file, and type BABEL as the prefix to it. Finally, a library is needed. Click Add library and add the bpmn2babel.asm file. Note that this file is auto-generated by Eclipse ATL when it compiles the bpmn2babel.atl file.

[3] **http://www.eclipse.org/modeling/m2m/downloads/index.php?project=atl**
[4] **http://www.bpm.scitech.qut.edu.au/research/projects/oldprojects/babel/tools/.**

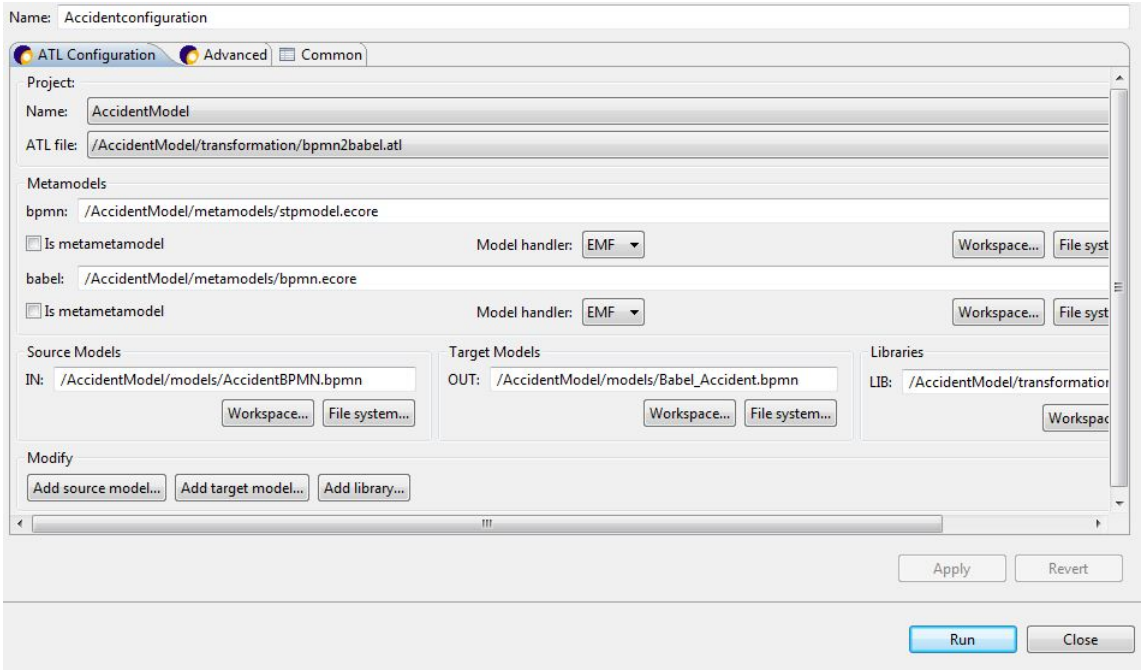


Figure 4.3: Configuring the ATL launch configuration.

3. *Transformation from BABEL BPMN to BPEL.* From the main window of the framework, (Fig. 4.2), choose the *"Convert BPMN to BPEL"* option. A new window pop-ups (Fig. 4.4), prompting the user to choose the BPMN file (generated in the previous step) and the corresponding WSDL file. Finally, click the *"Convert"* option in order to generate the abstract BPEL process. This last step consists of two procedures:

   (a) Firstly, the program edits the generated BPMN file as it consists of two <babelBpmn:Do children element (only one is needed) and then evolves the BABEL BPMN2BPEL library, in order to generate a temporary version of the BPEL file.

   (b) The BPEL file is used by the bpmn2bpel project, provided by our framework, in order to complete the BPEL activities.

   In this step, we parse the WSDL file, obtaining:

   - The namespace.
   - The process name.
   - The partnerLinkTypes, with the corresponding role and portType.
   - For each portType, the corresponding operation and input messages.

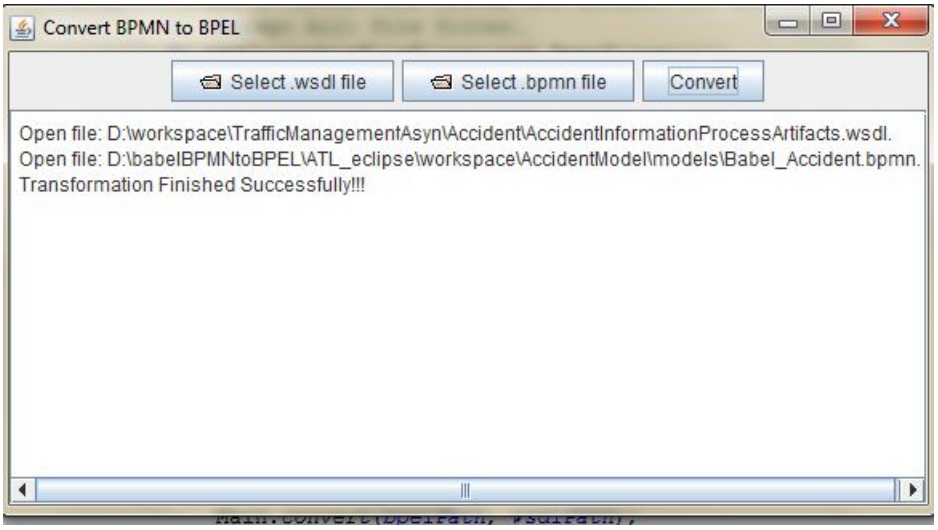


Figure 4.4: Transformation from BPMN to BPEL window.

- The properties, needed for creating the correlation sets at the BPEL file.

In order for this step to proceed with success, the name of the one role must contain the word *'Provider'* and the name of the other role must contain the word *'Requester'*. This restriction exists in order to be able to map correctly the roles to the corresponding 'myRole' and 'partnerRole' of the BPEL process.

Finally, we parse the BPEL file, for making the required changes (Fig. 4.5):

- Adapt the process definition.
- Add the partnerLinks, with the corresponding roles.
- Add the variables. Create one variable for each input message.
- Add the correlation sets.
- Adjust activities, such as 'receive', 'invoke', 'onMessage'.

In this step, we map the name of the invoke or onMessage activity (created with BABEL library) with an operation of the WSDL file. In case, there is not an operation with this name , we replace the invoke activity with an empty activity or for the onMessage activity we inform the user for the error and proceed.

Figure 4.5: Differences between BABEL BPEL output and bpmn2bpel output.

4.2 Transform OWL-Q to Astro's Monitoring Language

For the purpose of our implementation, we choose to use OWL-Q [34] for the definition of the QoS description of the web services.

Although OWL-S [40] provides a complete ontology for the description of Web services, it leaves the definition of QoS aspects to the service provider. OWL-Q is a semantic, rich and extensible QoS description model for Web services. It is designed modularly, incorporating several independent facets, each one focusing on a particular part of QoS-based Web service description. There are facets regarding the connection of OWL-Q with OWL-S, the description of QoS offers and requests, the QoS metric model, the definition of constraints, to name a few. OWL-Q also incorporates reasoning by allowing the definition of rules to reason about class properties.

Different facets are used for our implementation. The QoSSpec class (belogs to the Basic Facet) represents the actual QoS description of a WS. It describes the security and transaction protocols used, the cost of using the service and the associated currency for

the cost, the validity period of the offer or demand and an arbitrary OpenMath expression (om:OMOBJ). This expression represents what is or must be guaranteed and contains variables which are associated to QoS Metrics.

The QoS Metric Facet describes all the appropriate classes and properties used for a proper formal definition of a QoS metric. This metric facet is actually an upper ontology representing any abstract QoS metric. A specific QoS metric can be created by refining the QoSMetric class. The QoSMetric is one of the most important classes of OWL-Q representing a QoS metric. The values of a QoS metric are provided by a service provider or a requester or a third-party. It measures a QoSProperty on a specific ServiceElement. The value type of a QoSMetric is an instance of the QoSValueType class while the unit of the value is an instance of the Unit class. It can be a simple QoS metric measuredBy a MeasurementDirective or a complex one. ComplexMetrics are derived from other metrics with the help of a OMFunction.

The Function Facet describes all the appropriate concepts and properties for the proper definition of metric functions. The OMFunction class is the basic concept that represents a QoS Metric Function.

The Measurement Directive Facet describes the concept of measurement directive which is used for the measurement of simple metrics. A MeasurementDirective is composed of a URI that describes where and how to get a value of a resource's property.

The main structure of the OWL-Q classes used by our work is presented in Fig. 4.6.

4.2.1 Implementation Procedure

In this section, we analyze the procedure that must take place, in order to produce the monitoring properties. Two steps are required:

1. Define the OWL-Q file, specifying the QoS description, based on the structure depicted in Fig. 4.6.

2. From the main window of the framework, (Fig. 4.2), choose the *"Get the monitoring properties"* option. A new window pop-ups (Fig. 4.7), prompting the user to choose the choreography file (in which the monitoring properties will be added)) and the .owl file.

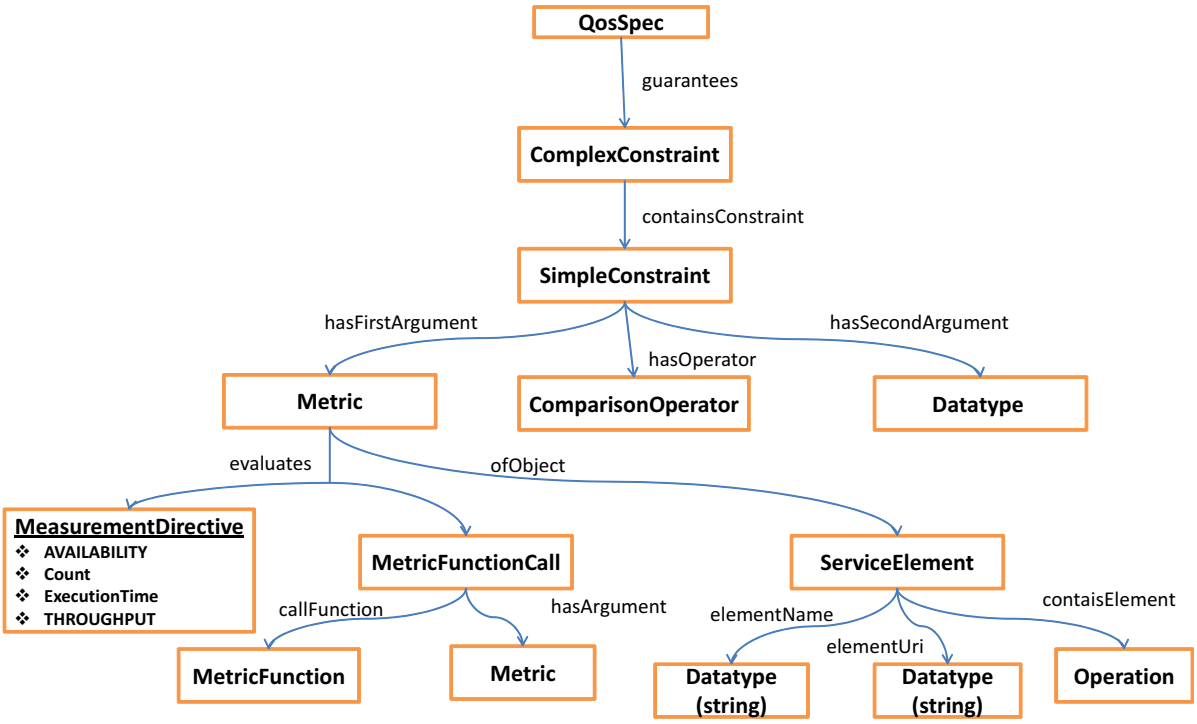


Figure 4.6: Structure of OWL-Q classes.

In order to parse the .owl file, we used the Pellet Reasoner [5]. The first step is to check the consistency of the model and then retrieve the required properties and classes needed to create the monitoring properties. By using the obtained properties and classes, we create Astro's Monitoring properties (if it exists), otherwise we check if the property can be monitored by the *"Extended Monitoring Framework"*. If this case, we forward it to the framework, otherwise the property is rejected.

The 'Extended Monitoring Framework', mostly relies on the results of Astro's monitor. It is used to evaluate properties like maximum or minimum execution time, throughput (minimum,maximum,average), but it can also be used to monitor the availability of a service. In order to start it, click on the *'Start Monitoring'*, before the execution of BPEL process and after the termination of the processes click to *'Stop Monitoring'*. This option terminates the execution of the monitors and also opens the file with the violated properties.

[5] http://clarkparsia.com/pellet/

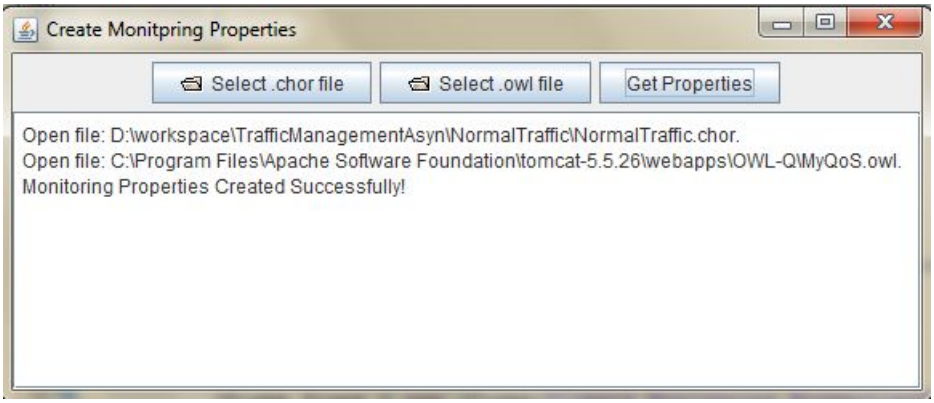


Figure 4.7: Creation of Monitoring Properties

Finally, when the properties are created, they are automatically added to the choreography file.

4.3 Astro's Monitoring Tool

As mentioned at the previous sections, the monitoring tool used in our implementation is Astro's WS-MON [6]. Astro is a research project in the field of web services and service-oriented applications, with a focus on the integration of business processes.

The tool provides the ability to monitor service compositions implemented in BPEL4WS. The tool has the following main features:

- The monitor engine and the BPEL execution engine are executed in parallel on the same application server. This allows for an integration of the two engines, still maintaining the two run-time environments distinct, and keeping the monitors clearly separated from the BPEL processes. Monitors observe their behavior by intercepting the input/output messages that are received/sent by the processes, and signal misbehaviors or, more in general, situations or events of interest.

- Both instance and class monitors are supported. *Instance monitors* deal with the execution of a single instance of BPEL business process, while *class monitors* extract information from and/or check the behavior of all the individual instances of a business process.

[6] http://astroproject.org/

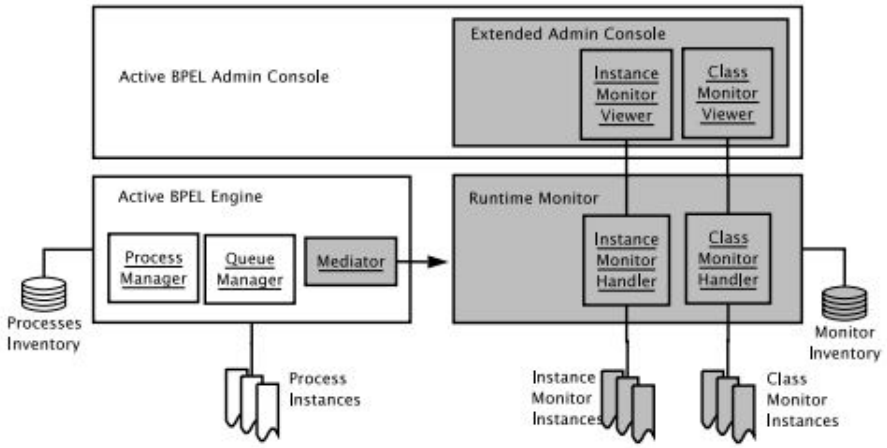


Figure 4.8: The ActiveBPEL engine extended with the run-time monitor environment

- An expressive monitoring language, namely Run-Time Monitoring specification Language (RTML) is proposed. It allows for expressing behavioral properties of service composition instances, together with the timing and statistical information and of composition classes.

- A technique for the automatic generation of the code implementing the instance and class monitors is provided. Monitors are automatically generated as Java programs that can be deployed in the run-time environment of the monitor engine.

At the following sections a more detailed survey of the tool is provided.

4.3.1 BPEL Execution Environment

Among the existing engines, the authors choose ActiveEndpoint's engine, ActiveBPEL [27], since it is available in an open source and also it implements a modular architecture that is easy to extend.

From a high-level point of view, the ActiveBPEL runtime environment can be seen as composed of four parts (light components of Figure 4.8). A Process Inventory contains all the BPEL processes deployed on the engine. A set of Process Instances consists of the instances of BPEL processes that are currently in execution. The BPEL Engine is the most complex part of the run-time environment, and consists of different modules (including the Process Manager, and the Queue Manager), which are responsible of the different

aspects of the execution of the BPEL processes. The Admin Console provides web pages for checking and controlling the status of the engine and of the process instances.

For the monitoring purposes, the most relevant aspects in the execution of BPEL processes are the creation and the termination of a new process instance, and the input and output of messages. The engine manages to create a new instance for a BPEL process in the inventory when one of its start activities is triggered by an incoming message. The creation of the process instance is supervised by the Process Manager, and consists in the activation of the initial set of BPEL activities for that process. When all the activities of a process instance have been executed, the Process Manager terminates that instance. The Queue Manager is responsible for dispatching incoming and outgoing messages. When an incoming message is received, the engine tries to find an active process instance that matches the correlation data included in the message. If such an instance is found, then the message is stored in the 'inbound queue' for that process instance, where it waits until it gets consumed by one of the activities of the process instance. If no matching instance is found, the message is parked in a unmatched message queue until a matching instance is found. The management of outgoing messages is much simpler. The engine provides an 'outbound queue', where outgoing messages are stored by invocation or reply activities. The Queue Manager is responsible for picking messages from the 'outbound queue' and for dispatching them to the destination services.

4.3.2 Run-Time Monitoring Environment

The Astro's WS-Mon is implemented as an extension of the ActiveBpel environment. In particular, the ActiveBpel is extended with five new components (dark part of Fig. 4.8). The *Monitor Inventory* and the *Monitor Instances* are the counterparts of the corresponding components of the BPEL engine: the former contains all the monitors deployed in the engine, while the later is the set of instances of these monitors that are currently in execution. The *RunTime Monitor* (RTM) is responsible to support the life-cycle (creation and termination) and the evolution of the monitor instances. The *Mediator* allows the RTM to interact with the Queue Manager and the Process Manager of the BPEL engine and to intercept input/output messages as well as other relevant events such as the creation and termination of process instances. The *Extended Admin*

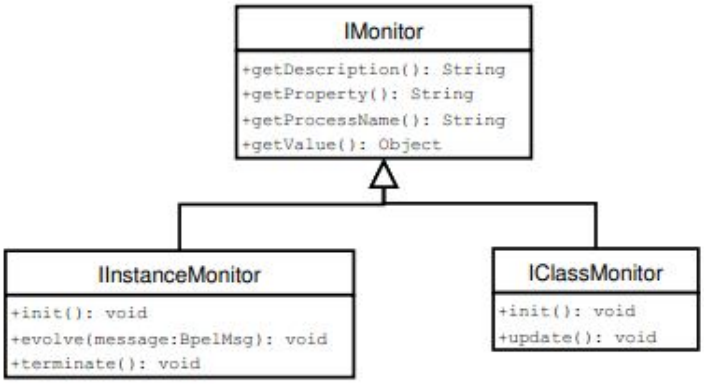


Figure 4.9: Methods of a monitor Java class.

Console is an extension of the ActiveBPEL Admin Console that presents, along with other information on the BPEL processes, the information on the status of the corresponding monitors.

The framework supports two kinds of monitors. *Instance Monitors* (IMs), which observe the execution of a single instance of a BPEL process; and *Class Monitors* (CMs), which report aggregated information on all the instances of a given BPEL process. The two kinds of monitors are reflected in the architecture of the monitoring framework. Indeed, according to 4.8, there are two distinct sets of monitor instances, namely IM Instances and CM Instances. In the RTM, the IMs and CMs are managed by two specific handlers, the IM Handler and the CM Handler. Also the Extended Admin Console provides two viewers, a IM Viewer and CM Viewer, to display the status of the two kinds of monitors.

4.3.3 Structure of Monitors

A monitor is a Java class implementing the IMonitor interface, described in Fig. 4.9. More precisely, IMs implement the IInstanceMonitor interface, while CMs implement the IClassMonitor interface.The IMonitor interface defines four methods which are common to all monitors: **getProperty** and **getDescription** return a short and a long description of the property that is monitored; **getProcessName** returns the name of the BPEL process the monitors are associated to and **getValue** returns the current value of the monitor.

The methods defined by interfaces IInstanceMonitor and IClassMonitor manage the

evolution of the monitors, and are better explained describing the life-cycle of instance and class monitors. The IMs life-cycle is influenced by three events: the process instance creation,the input/output of messages, and the termination of the process instance. When the RTM receives the notification of the creation of a new BPEL process,it creates a set of monitor instances that are specific for that process instance.The monitor instances are initialized through the method **init**. When the RTM receives a message from the Mediator, it sends it to the Instance Monitor Handler which dispatches the message to all the matching monitor instances through the method **evolve**. For each message, the Mediator provides also information on the process instance receiving/sending the message, as well as on the BPEL process specification corresponding to the instance.The process termination is captured via a termination event, which is dispatched, through the invocation of method terminate, to all the monitor instances associated to the process instance.

The life-cycle of a class monitor is quite different. Method **init** is called only once, when the single instance of the class monitor is created.The evolution of the class monitor is triggered whenever the RTM receives a message or an event is received from any instance of the BPEL process to be monitored. More precisely, after the Instance Monitor Handler has dispatched the event or message to the relevant monitor instance, and this has been updated, it signals to the Class Monitor Handler that also the class monitors have to be updated.The Class Monitor Handler invokes method update on all the different class monitors associated to the BPEL process, which can update their internal status.

4.3.4 Monitoring Language

RTML, the Run-Time Monitoring specification language is used to define the monitoring properties. RTML is rather expressive: it allows for the specification of IMs as well as CMs; moreover, it allows for specifying boolean properties related to the execution of processes, as well as statistic properties and time-related properties.

According to the framework the relevant events for monitors are:

- The creation and termination of a process instance; these two events are modeled through keywords ‘start’ and ‘end’ in RTML.

- The input and output of messages; in this case, RTML requires to specify the link on which the message is received or sent, the fact that the message is an input or an output, and the message type.

In some cases, it is preferable to speak of the effects of an event on the status of an interaction protocol, rather than of the event itself. In order to express such cases in RTML we can use: 'cause(link.var = val)' to denote all events that cause variable var of BPEL process link to assume value val.

The complete grammar for events e is the following:

e ::= start | end |

msg(link.input/output = msg[opt-constraints]) |

cause(link.var = val) |

cause(link.state = label)

Instance Monitor Formulas

The following grammar defines the formulas that specify instance monitors. We distinguish boolean formulas b, which monitor properties that can be either true or false, and numeric formulas n, which monitor properties that define a numerical value.

b ::= e | Yb | O b | H b | b Sb |

n = n | n > n | ¬ b | b ∧ b | true

n ::= count(b) | time (b) | b?n:n |

n + n | n - n | n * n | n/n | 0 | 1 | ...

A *boolean formula* can be an event e, a Past LTL [26] formula (operators Y, O, H, and S), a comparison between numeric formulas, or a logic combination of other boolean formulas. A *numeric formula* can be either a counting formula (operators count and time), a conditional expression (b?n:n), or an arithmetic operation on other numeric formulas.

Formulas are evaluated whenever a relevant event is received by the instance monitor. Formula e is true if it is compatible with the occurring event. Past LTL formulas have the following meaning:

- Y b means b was true in the previous step;
- O b means b was true (at least) once in the past;
- H b means b was true always in the past;

- b1 S b2 means b1 has been true since b2.

The automatic translation of an RTML instance formula into the Java code implementing the monitors is provided by the framework.

Class Monitor Formulas

Also in case of class monitors, boolean formulas B and numeric formulas N are provided, as shown by the following grammar.

B ::= And (b) | Y B | O B | H B | B S B |

N = N | N >N| ¬B | B ∧ B | true

N ::= Count(b) | Sum (n) |

N + N | N - N | N * N | N/N | 0| 1 | ...

where b and n are instance monitor formulas. Most of the operators are identical to those of the instance monitor formulas. Boolean formula And (b) checks if property b is true for all the instances of the BPEL process corresponding to the monitor. Numeric formula Count(b), instead, counts the number of instances of the BPEL process for which formula b holds. Numeric formula Sum (n) is similar, but aggregates numeric instance module formulas: it sums up the values of numeric formula n on all the instances of the BPEL process.

Also in case of class monitors, a translator from RTML to Java code is provided. The key difference with respect to instance monitors is in the implementation of operators And, Count, and Sum. Indeed, these operators serve as link between class-level monitoring and instance level monitoring. The translation algorithm adopts the following approach:

- An instance monitor is generated for each instance property b or n that appears as argument of operators And, Count, and Sum in the class formula.
- One class monitor is generated that aggregates the data of the instance monitors according to the formula.

4.3.5 Implementation Issues

As mentioned at the previous sections, Astro's WS-Mon takes as input a choreography file in order to generate the Java code responsible for the creation of the monitors for the composed process and deploys them to the monitor framework. The monitor generation

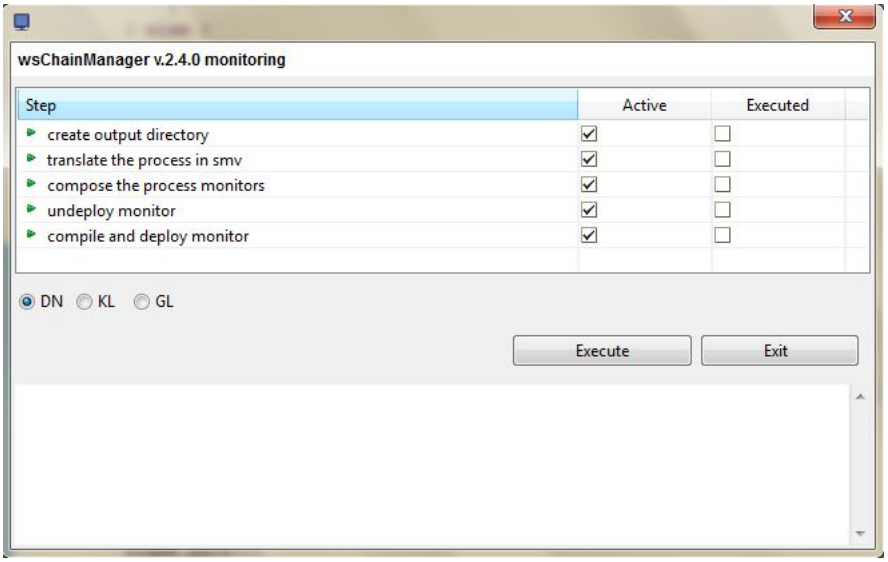


Figure 4.10: WS-mon front end.

process, has been implemented at the Astro framework, as an Eclipse plugin, presented in Fig. 4.10.

To present the status of the monitors associated with each process instance, Astro provides a WS-console which, extends the Active BPEL administration console(Fig. 4.11).

For the purpose of our implementation, we also need these results,to be written at files. In order to achieve this, at the main framework click *'Add code for Logs'* and select the 'build' folder that was created, during the monitoring generation process. Then, re-run the 'compile and deploy monitor' step, from Astro's plug-in. The file contains the following information:

- The *timestamp* of the event.
- The *process name*.
- The *functional layer*.
- The *property*.
- The *value* the property has.

These information are required in order to check violations of the monitored properties.

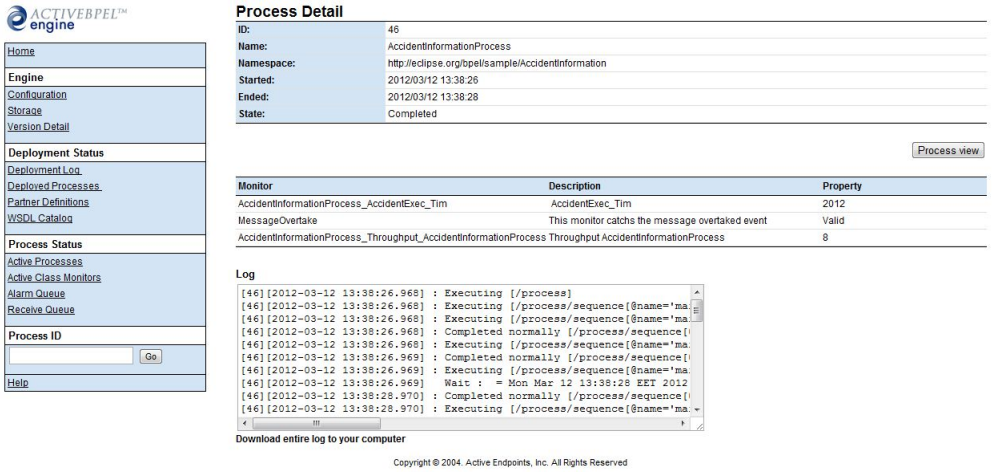


Figure 4.11: WS-console.

```
2012-03-15 13:40:37.551 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:40:37.551 AirPollution    AirPollution_AV_Exec    0.52275 >=  0.5
2012-03-15 13:41:41.792 CallService CallService_AV_Through  6.0 <   25.5
2012-03-15 13:41:41.792 CallService CallService_MAX_Throu   6.0 <=  20.0
2012-03-15 13:41:41.792 CallService CallService_Min_Through 6.0 <   9.0
2012-03-15 13:41:41.792 CallService CallService_Throughput  6.0 <   10.0
2012-03-15 13:41:53.211 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:42:05.052 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:42:16.924 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:42:39.45  NormalTraffic   NormalTrafficExec_Time  20.311  >=  20.03
2012-03-15 13:42:55.518 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:43:11.867 CallService CallService_AV_Through  12.0    <   25.5
2012-03-15 13:43:11.867 CallService CallService_MAX_Throu   18.0    <=  20.0
2012-03-15 13:43:11.867 CallService CallService_Min_Through 6.0 <   9.0
```

Figure 4.12: Example of violated properties

4.4 Report Violations

The last part of our implementation reports the occurred violations of the monitoring properties. A snapshot of the file is depicted at Fig. 4.12

The file consists of the following information:

- The *timestamp* of the event (when the violated event occurred).
- The *process name*.
- The *property*.
- The *value* the property has.

- The comparison operator used to check the property
- The desired upper/lower bound the property has.

Chapter 5

Case Study

This chapter describes the case study we decided to use in order to test our monitoring framework.

5.1 Traffic Management Case Study

This case study describes a traffic management system designed to manage normal traffic situations as well as emergency cases [25]. Such emergency case handling includes several different actions, such as directing the rescue forces to the accident location and managing traffic deviations. Fig. 5.1 and Fig. 5.2 respectively illustrate these two cases. Each figure depicts the three functional layers presented at chapter 1. In both cases, workflow tasks are executed either manually or by mapping them on (Web) services. Each service is then mapped to the appropriate infrastructure.

The actors involved are *traffic managers*, i.e., individuals accountable for entities controlling the traffic management system, generic *rescue forces* (e.g., police and ambulances), and *citizens*, such as motorists and pedestrians.

Fig. 5.1 illustrates normal traffic conditions, where the system tries to optimize some parameters such as total noise, overall throughput, and air pollution. In particular, the system shall consider different needs, such as the ones of pedestrians and motorists, and other factors like heavy traffic, public events, school and working hours, holidays or public regulations which may alter traffic demand and needs during conditions that do not involve emergencies. The system interrupts the Normal Traffic Situation process, when

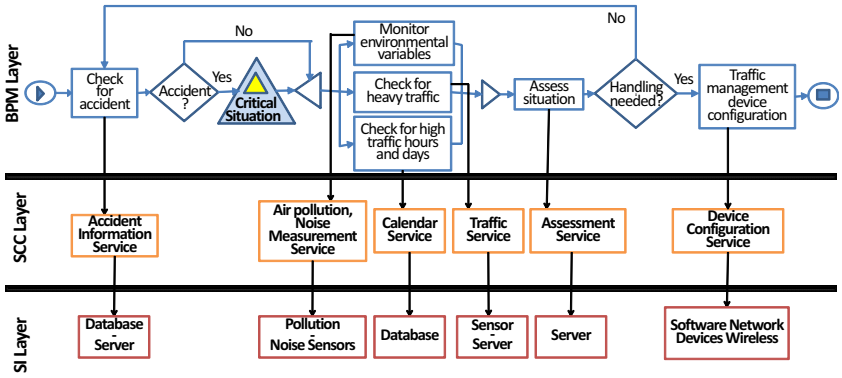


Figure 5.1: Normal Traffic Situation

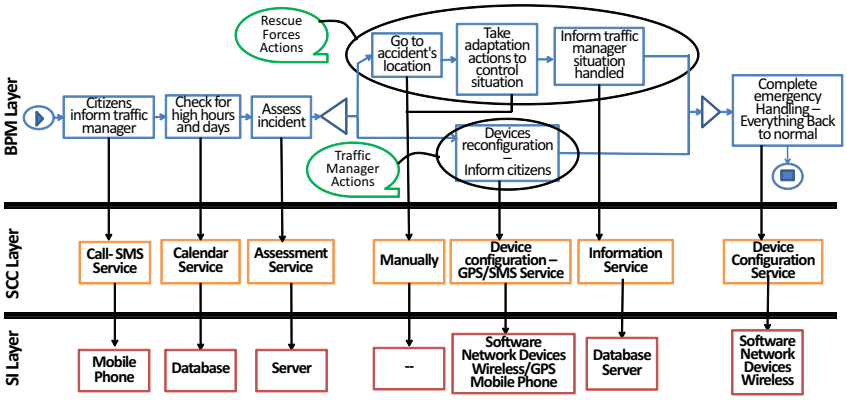


Figure 5.2: Critical Traffic Situation

an accident happens, and jumps to the Critical Traffic Situation subprocess.

Fig. 5.2 depicts a Critical Traffic Situation, in which a serious car accident occurs at a central road. In particular, the involved citizens inform the traffic manager that must control the overall traffic situation (control traffic devices, inform citizens) and assess the incident so as to inform the appropriate rescue forces about the accident and direct them to the specific location. Moreover, the traffic manager monitors the environment variables, such as air pollution and noise. Different adaptation actions must be taken by the traffic manager as well as by the rescue forces, such as:

- Traffic management device reconfiguration (e.g., traffic lights) by the traffic manager, in order to reduce stop-and-go traffic. This should also help to keep air pollution low, even if it is not critical during emergency situations.

- Accident reporting to citizens via their devices (e.g, GPS, mobile phones) by the

traffic manager to avoid traffic congestion at the accident location.

- Traffic closing/limiting to or from the involved location by the rescue forces.
- Traffic deviation by the rescue forces through alternative places not intended for heavy traffic.

After a complete emergency handling, there is a gradual return back to the normal situation. The rescue forces inform the traffic manager, who updates the system and informs the citizens through their devices.

5.2 Case Study Implementation

In Section 5.1 we presented the main idea of the case study, without including details of our implementation. In this section we introduce in detail our model and also the services we have implemented in order to cover the needs of our model. Both services and main model were implemented in BPEL4WS as asynchronous web services.

5.2.1 Accident Information Service

This service is responsible to inform the traffic manager for the occurrence of an accident. The input of this process is the city we want to examine and also a random integer number that is provided by the user. An accident occurs when the remainder of the division of the integer with number nine is equal to eight. This means that the possibility of an accident is 89%. In case of an accident the process selects randomly the location of the accident, otherwise it informs the traffic manager that no accident has been occurred. The output of this process is either the location of the accident or the message "No accident".

5.2.2 Air Pollution Service

The aim of this service is to "measure" the air pollution of a particular city. The scale used to define the air pollution metrics is the one used in Canada [28]. According to this scale :

- 1-3 Air Quality Health Index: Low health risk.

- 4-6 Air Quality Health Index: Moderate health risk.
- 7-10 Air Quality Health Index: High health risk.
- Above 10 Air Quality Health Index: Very High health risk.

The input of this process is the city we want to examine and also a random integer number that the user provides. The result of the remainder of the division of the integer with number eleven, indicates the health risk. For example, if the result is equal to 5, then the health risk is "Moderate". The area of the city that the air pollution service checks is picked randomly. The output of this service is the health risk and also the location that have been checked.

5.2.3 Noise Measurement Service

Respectively to the aforementioned service, noise measurement service is responsible to "check" the noise pollution in a particular city. The scale used in this service has three levels: Normal, High and Very High. Noise is measured in decibel (db) [1]. According to these levels:

- 1-65 db : Normal noise level.
- 66-85 db : High noise level.
- 86-100 db: Very High noise level.

The input of this process is the city we want to examine and also a random integer number that the user provides. The result of the remainder of the division of the integer with 100, indicates the noise level. For example, if the input number is equal to "997" then the result is equal to "97". According to the above levels the noise level is "Very high". The output of this service is the noise level and the area of the city that has been randomly picked.

[1] http://en.wikipedia.org/wiki/Decibel

5.2.4 Calendar Service

Calendar service is responsible to check for public events, school and working hours, holidays or public regulations which may alter traffic demand and needs. The initial aim of this process was to check the current time and day in order to decide if it is rush hour or holiday. Unfortunately, BPEL 1.0 did not provide a function to get current time and day, that's why we ask the user for a random integer number in order to decide the traffic level. The possibility to be a holiday is 10%. In this case the traffic level is "Low". The possibility to be rush time is 30% and in this case the traffic level is "Heavy". The possibility to be night is also 30% and in this case the traffic level is equal to "Low". Finally, in all the other cases (30% possibility) the traffic level is "Normal".

In case the traffic level is equal to "Heavy", the calendar service also returns the location.The location is decided randomly and the process may return one to three locations.

The input of this process is also the city we want to examine and a random integer number. Finally the output, is the traffic level and the location.

5.2.5 Assessment Service

The Assessment web service is responsible to decide the actions that must be taken according to the result of the last three aforementioned web services. The adaptation actions that can take place are the configuration of traffic lights, the placement of traffic policeman or both. We examine three different cases:

- First Case. The air pollution level is "Low" or "Moderate", the noise pollution level is "Normal" and the heavy traffic level is "Low" or "Normal". In this case the assessment service takes no adaptation actions.

- Second Case. The air pollution level is "High" or "Very High", the noise pollution level is "High" or "Very High" and the heavy traffic level is "Heavy". In this case the assessment service takes both adaptation actions.

- Third Case. One or two of the above levels has exceeded the "Normal" value. In this case, we decide randomly which adaptation action should take place.Only one of them is going to be executed.

The output of this process is the adaptation action that should take place and the location.

5.2.6 Device Configuration Service

The Device Configuration service is responsible for the configuration or re-configuration of traffic lights. The input of the process is the action (configuration or re-configuration) that should take place and the area of the city. Depending on the action the service "executes" the appropriate activity. The output is a report that informs the traffic manager that the actions executed with success.

5.2.7 Call Service

The aim of this process is to "inform" the appropriate role for the existence of an event. This services "executes" an activity that indicates that a "call" is performed.

5.2.8 Incident Assessment

This process is only executed in case of an accident. The aim of this process is to check for heavy traffic at the location of the accident. It invokes the calendar service, in order to get the locations of heavy traffic and then examines whether the accident's place is included. If this is the case, this information is passed to the output. The input of this service is the location of the accident, the involved city and also a random integer number needed for the calendar service.

5.2.9 Implementation of Main Model

The aim of this main process is to handle the normal traffic situation as well as the case of an accident. The input of this process is the city to be checked and a random integer needed by the aforementioned web services. Unfortunately BPEL 1.0 does not have a function for the generation of random numbers, so the user should provide this number. Initially our model is applied only for "Heraklion" city, but if needed is easily extended. In case the user types another city, the system returns an "invalid city" message and the process ends. Otherwise, the process invokes the Accident Information service and

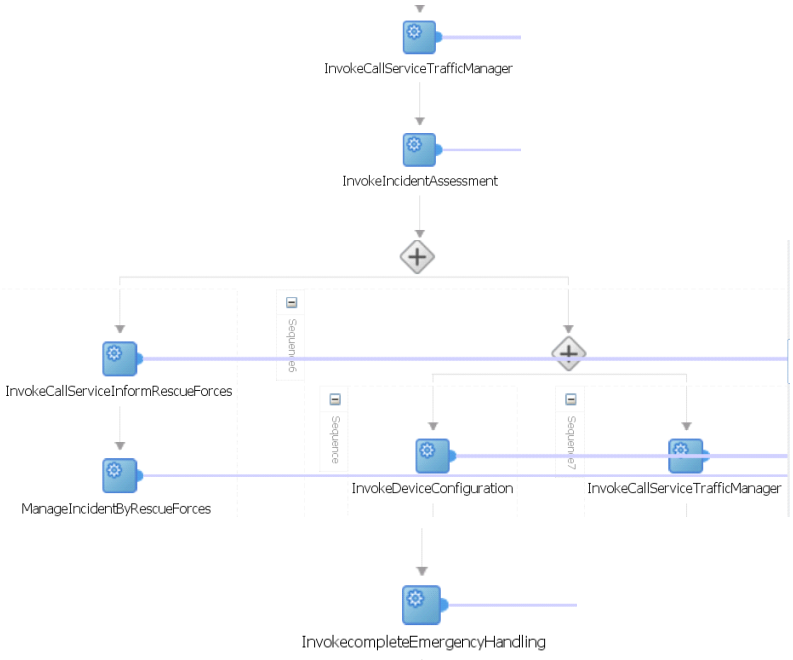


Figure 5.3: Critical Situation Activities

the output is checked in order to decide what actions should take place (jump to critical situation or execute normal's situation activities).

5.2.9.1 Critical Traffic Situation

The process interrupts the normal traffic situation in order to handle the accident. Figure 5.3 illustrates the activities that are executed in this case. The call service is invoked in order to inform the traffic manager about the accident. Then the Incident Assessment service informs both the traffic manager and the rescue forces, for traffic at the accident's location.

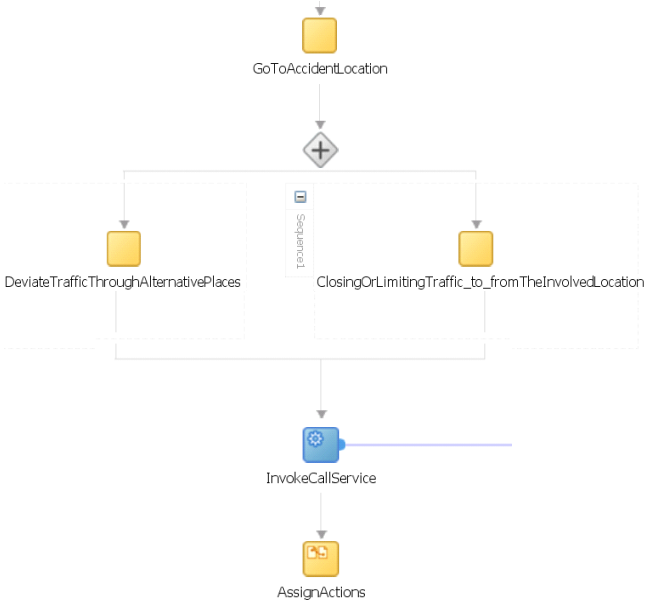


Figure 5.4: Manage Incident By Rescue Forces.

The traffic manager invokes the Device Configuration service, in order to reduce stop-and-go traffic, and the Call service, in order to inform citizens about the accident,avoiding thus traffic congestion.

On the other hand, the rescue forces are informed about the accident (Call service is invoked) and the ManageIncidentByRescueForces subprocess is invoked (Figure 5.4). Aim of this subprocess is to execute all the actions that the rescue forces should do, in order to handle the accident. Most of these are manually activities, so in our subprocess are implemented by an empty BPEL activity. The rescue forces go the accident's locations in order to close/limit the traffic and also to deviate it through alternative places not intended for heavy traffic. When the situation is handled the Call service is invoked in order to inform the traffic manager.

Once the two subprocesses are executed, actions for the completion of the emergency handling takes place (Figure 5.5). They are responsible to reinstate everything back to normal. The traffic manager invokes the Device Configuration service but in this case the input is the re-configuration of the traffic lights. On the other hand, the rescue forces fix traffic back to normal. This is also depicted with an empty BPEL activity, as it is manual. Finally the rescue forces inform the traffic manager that situation is handled and in turn traffic manager informs citizens. In both cases, the call service is invoked.

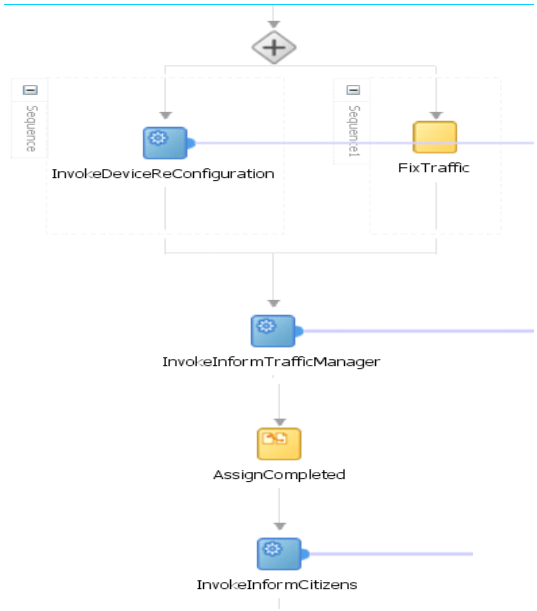


Figure 5.5: Complete Emergency Handling.

5.2.9.2 Normal Traffic Situation

Normal traffic activities are executed either after the handling of an accident or after the invocation of Accident Information service, in case no accident has occurred. The process checks the environmental variables and the traffic. For this reason the Noise Measurement, the Air Pollution and the Calendar services are invoked.

Then the Assessment service is invoked in order to determine which adaptation actions should take place. Depending on this result different services are invoked (Figure 5.6). Three different cases are detected:

- First Case. The Assessment's service output is "No Actions", so there is no need to take any adaptation actions.
- Second Case. According to the result, the system must configure the traffic lights and also place traffic policeman. In order for these two requirements to be fulfilled, the Device configuration and the Call services must be invoked.
- Third Case. One of the above adaptation actions must take place. The process checks the output and if it is equal to "Configure traffic lights", the Device Configuration service is invoked. Otherwise, the Call service is used, in order to inform the traffic policeman.

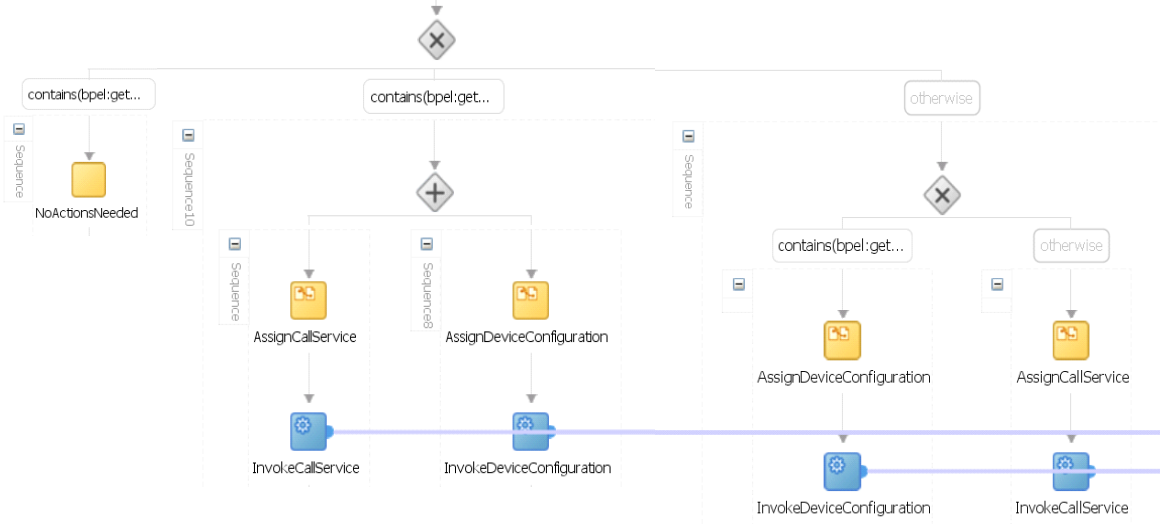


Figure 5.6: Take Adaptation Actions.

In case the Device Configuration service, is called the process invokes this service again, with a different parameter. This time the input of the service is "Re-Configuration", instead of "Configuration".

Finally the process returns a report, that informs the user, whether there was an accident and which adaptation actions took place.

5.3 Monitoring Properties

For the implementation of our Case Study, we monitor the following monitoring properties of each process:

- Execution Time for each process.
- Minimum, Maximum and Average Execution Time.
- Throughput.
- Minimum, Maximum and Average Throughput.
- Successability [2].
- Availability.

[2]http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsqm

- Minimum, Maximum and Average Availability.
- Number of times no accident has occurred.
- Number of times high environmental conditions has occurred (traffic, air pollution, noise pollution).

Table 5.1 presents for each aforementioned property, which monitoring component is responsible for its evaluation and also distinguishes the properties that refer to instance monitors from those that refer to class monitors.

Monitoring Property		Monitoring Tool		Instance Monitor	Class Monitor
		Astro	Extended Framework		
Execution Time		✓		✓	
Minimum Execution Time			✓		✓
Maximum Execution Time			✓		✓
Average Execution Time		✓			✓
Throughput			✓		✓
Minimum	Throughput		✓		✓
Maximum					
Average					
Successability		✓			✓
Availability			✓		✓
Minimum	Availability		✓		✓
Maximum					
Average					
Times of no accident, environmental conditions		✓			✓

Table 5.1: Monitoring Properties.

In order to define the aforementioned monitoring properties, we based to the following definitions :

- **Execution Time** is the time taken by a Web Service to process its sequence of activities.
- **Throughput** represents the number of Web Services requests served in a given time period. In order to cover the needs of our implementation , the time period used is

equal to one minute and thirty seconds.

- **Successability** can be calculated as the number of successful invocations over the number of invocations.

- **Availability** is the absence of service downtimes. Availability represents the probability that a service is available. In order to to monitor this property, the Extended Monitoring Component pings the service every fifteen seconds and after two minutes and fifty seconds reports the probability that a service is available.

5.3.1 Definition of Monitoring Properties in RTML

Table 5.2 depicts the definition of the above monitoring properties at RTML, the monitoring language Astro use's. These properties arise automatically with the transformation of the OWL-Q ontology.

Property	Property expressed in RTML
Execution Time	time(!end S start)
Successability	(Count(O end)) / (Count(O start))
Number of High Traffic	Count (cause(Calendar.pc=CalendarService_HeavyTraffic))
Number of No Accidents	Count (cause(AccidentInformationProcess.pc=NoAccident))
Number of High-Very Air Pollution	Count (cause(AirPollution.pc=AirPollution_VeryHighRisk)) + Count (cause(AirPollution.pc=AirPollution_HighRisk))
Number of High-Very Noise Pollution	Count (cause(NoiseMeasure.pc=NoiseMeasure_VeryHighTraffic)) + Count (cause(NoiseMeasure.pc=NoiseMeasure_HighTraffic))

Table 5.2: Definition of Monitoring Properties at RTML.

Chapter 6

Experimental Evaluation

In this section, we experimentally measure the above monitoring properties, for our case study. We execute our scenario for 5 times and each time, we execute forty iterations with a different random number as input for each loop. In order to create this number, we multiply, every time, the last integer that has been created with the number of the loop we execute.

For the execution of our case study, we use soapUI eclipse-plugin [1]. soapUI is an Open Source Functional Testing Tool, mainly it is used for Web Service Testing . SoapUI supports multiple protocols such as SOAP, REST, HTTP, JMS, AMF and JDBC. It enables one to create advanced Performance Tests very quickly and run Automated Functional Tests.

All experiments were carried out in a PC with processor Intel Core2 Duo 2,00 GHz, 3 GB Ram, running Windows 7 64-bit. Also, time was measured in milliseconds.

6.1 Metrics Input

In Tables 6.1 and 6.2, we provide the upper or lower bound we have used in order to test which monitoring properties are reported as violated during the execution of the case study. A violation occurs when a monitoring value exceeds the threshold we have defined.

The above values are selected because we want violation to be occurred during the execution of the process. This choice was made in order to test that our monitoring

[1] http://www.soapui.org/IDE-Plugins/eclipse-plugin.html

Process	Execution Time					Throughput				
	Min (sec)	Max (sec)	Average (sec)	Class (sec)	Operator	Min	Max	Average	Class	Operator
AccidentInformationProcess	2.1	2.5	2.5	2.1		4	7	6.5	4	
AirPollution	1	1.1	0.5	1.03		4	7	6.3	4	
Assesement	2.03	2.29	2.1	2.1		4	8	6.1	5	
CallService	1.1	1.5	1.1	1.1		9	20	25.5	10	
Calendar	1.05	1.2	1.09	1.1		7	8	9.9	8	'<'
DeviceConfiguration	1	1.04	1	1.03	'>='	3	5	3.0	4	OR
IncidentAssessment	1.1	1.3	1.1	1.2		1	3	1.5	2	'=<'
InformationService	5.5	6	5.2	5.2		1	3	1.5	2	
NoiseMeasure	1	2	0.9	1.5		4	6	6	4	
NormalTraffic	18	21	20	20.03		4	8	6.3	5	

Table 6.1: Thresholds of Monitoring Properties.

Process	Count No Accidents		Count of High Traffic		Count of High Noise		Count of High Pollution		Availability					Successability	
	Class	Oper-ator	Class	Oper-ator	Class	Oper-ator	Class	Opera-tor	Min	Max	Average	Class	Oper-ator	Class	Oper-ator
AccidentInformation Process	50	'>='							0.99	0.99	0.99	0.99		0.9	
AirPollution							50	'>='	0.99	0.99	0.99	0.99		0.9	
Assesement									0.99	0.99	0.99	0.99		0.9	
CallService									0.99	0.99	0.99	0.99		0.9	
Calendar			50	'>='					0.99	0.99	0.99	0.99	'<='	0.9	'=<'
DeviceConfiguration									0.99	0.99	0.99	0.99	OR	0.9	
IncidentAssessment									0.99	0.99	0.99	0.99	'<'	0.9	
InformationService									0.99	0.99	0.99	0.99		0.9	
NoiseMeasure					50	'>='			0.99	0.99	0.99	0.99		0.9	
NormalTraffic									0.99	0.99	0.99	0.99		0.9	

Table 6.2: Thresholds of Monitoring Properties.

framework catches the occurring violations at run-time.

6.2 Execution of the Scenario

As mentioned before, for the purpose of our evaluation, we have executed five times our scenario and at each execution a loop of forty iterations is used. During the executions, the occurred violations were only related to throughput (min,max,average,class) and to execution time (min,max,average,class). As expected availability and successabilty were never violated, since our services were always up.

6.2.1 First Execution

The random integer used as input for this execution is '9'. During this first execution **'253'** violations were reported and the first '45' are depicted in Figure 6.1.

During this execution, twenty times an accident did not occur,three times we had high noise pollution , twenty-seven times high air pollution and forty-two times heavy traffic. According to Table 6.2, none of these properties is violated.

6.2.2 Second Execution

The random integer used as input for the second execution is '8'. As mentioned, this number changes during each loop. During this second execution **'111'** violations were reported and the first '45' are depicted in Figure 6.2.

During the second execution, eighteen times an accident did not occur, one time we had high noise pollution , twenty-five times high air pollution and forty-six times heavy traffic. According to Table 6.2, none of these properties is violated.

6.2.3 Third Execution

The random integer used as input for the third execution is '25'. During this execution **'59'** violations were reported and the first '51' are depicted in Figure 6.3.

During this execution, twenty times an accident did not occur, we did not have high noise pollution , twenty-six times high air pollution occurred and forty times heavy traffic occurred. According to Table 6.2, none of these properties is violated.

```
2012-03-15 12:14:59.509 NoiseMeasure    NoiseMeasure_AV_Exec    1.123   >=  0.9
2012-03-15 12:14:59.502 NoiseMeasure    NoiseMeasure_Min_Exec   1.057   >=  1.0
2012-03-15 12:15:02.059 DeviceConfiguration DeviceConfigurationExecTime 1.076   >=  1.03
2012-03-15 12:15:02.065 DeviceConfiguration DeviceConfiguration_AV_Exec 1.069   >=  1.0
2012-03-15 12:15:02.099 DeviceConfiguration DeviceConfiguration_Max_Exec    1.076   >=  1.04
2012-03-15 12:15:02.212 DeviceConfiguration DeviceConfiguration_Max_Exec    1.076   >=  1.04
2012-03-15 12:15:02.103 DeviceConfiguration DeviceConfiguration_Min_Exec    1.076   >=  1.0
2012-03-15 12:15:05.399 AirPollution    AirPollution_AV_Exec    0.5525  >=  0.5
2012-03-15 12:15:08.677 DeviceConfiguration DeviceConfiguration_Max_Exec    1.076   >=  1.04
2012-03-15 12:15:08.805 DeviceConfiguration DeviceConfiguration_Max_Exec    1.076   >=  1.04
2012-03-15 12:15:11.991 AirPollution    AirPollution_AV_Exec    0.7103333000000001  >=  0.5
2012-03-15 12:15:16.251 CallService CallService_AV_Exec 1.975   >=  1.1
2012-03-15 12:15:16.239 CallService CallServiceExecTime 1.984   >=  1.1
2012-03-15 12:15:16.34  CallService CallService_Max_Exec    1.984   >=  1.5
2012-03-15 12:15:16.341 CallService CallService_Min_Exec    1.984   >=  1.1
2012-03-15 12:15:19.814 AirPollution    AirPollution_AV_Exec    0.53675 >=  0.5
2012-03-15 12:15:23.104 CallService CallService_AV_Exec 1.515   >=  1.1
2012-03-15 12:15:23.174 CallService CallService_Max_Exec    1.984   >=  1.5
2012-03-15 12:15:30.535 CallService CallService_AV_Exec 1.3463334   >=  1.1
2012-03-15 12:15:30.55  CallService CallService_Max_Exec    1.984   >=  1.5
2012-03-15 12:15:33.822 AirPollution    AirPollution_AV_Exec    0.5296667   >=  0.5
2012-03-15 12:15:37.018 CallService CallService_AV_Exec 1.262   >=  1.1
2012-03-15 12:15:37.091 CallService CallService_Max_Exec    1.984   >=  1.5
2012-03-15 12:15:44.512 CallService CallService_AV_Exec 1.211   >=  1.1
2012-03-15 12:15:44.529 CallService CallService_Max_Exec    1.984   >=  1.5
2012-03-15 12:15:47.781 AirPollution    AirPollution_AV_Exec    0.52625 >=  0.5
2012-03-15 12:15:50.956 CallService CallService_AV_Exec 1.1771666   >=  1.1
2012-03-15 12:15:50.966 CallService CallService_Max_Exec    1.984   >=  1.5
2012-03-15 12:15:57.459 Assesement  AssesementExecTime  2.135   >=  2.1
2012-03-15 12:15:58.611 CallService CallService_AV_Exec 1.1532858000000001  >=  1.1
2012-03-15 12:15:58.692 CallService CallService_Max_Exec    1.984   >=  1.5
2012-03-15 12:16:01.847 AirPollution    AirPollution_AV_Exec    0.5235  >=  0.5
2012-03-15 12:16:04.983 CallService CallService_AV_Exec 1.13475 >=  1.1
2012-03-15 12:16:05.015 CallService CallService_Max_Exec    1.984   >=  1.5
2012-03-15 12:16:14.506 AirPollution    AirPollution_AV_Exec    0.5211667   >=  0.5
2012-03-15 12:16:20.008 CallService CallService_AV_Through  8.0 <   25.5
2012-03-15 12:16:20.007 CallService CallService_MAX_Throu   8.0 <=  20.0
2012-03-15 12:16:20.005 CallService CallService_Min_Through 8.0 <   9.0
2012-03-15 12:16:20.004 CallService CallService_Throughput  8.0 <   10.0
2012-03-15 12:16:20.024 DeviceConfiguration DeviceConfiguration_MAX_Throu   4.0 <   5.0
2012-03-15 12:17:06.535 CallService CallService_AV_Exec 1.1206666   >=  1.1
2012-03-15 12:17:06.621 CallService CallService_Max_Exec    1.984   >=  1.5
2012-03-15 12:17:07.859 CallService CallService_AV_Exec 1.1094000000000002  >=  1.1
2012-03-15 12:17:07.858 CallService CallService_Max_Exec    1.984   >=  1.5
2012-03-15 12:17:14.102 CallService CallService_AV_Exec 1.1002726999999999  >=  1.1
```

Figure 6.1: Violations of First Execution.

```
2012-03-15 12:40:15.55  NoiseMeasure    NoiseMeasure_AV_Exec    1.039   >=  0.9
2012-03-15 12:40:15.646 NoiseMeasure    NoiseMeasure_Min_Exec   1.017   >=  1.0
2012-03-15 12:40:18.696 NormalTraffic   NormalTrafficExec_Time  20.252  >=  20.03
2012-03-15 12:40:18.702 NormalTraffic   Normal_AV_Exec  20.261  >=  20.0
2012-03-15 12:40:18.758 NormalTraffic   Normal_Min_Exec 20.252  >=  18.0
2012-03-15 12:40:34.902 AirPollution    AirPollution_AV_Exec    0.523   >=  0.5
2012-03-15 12:40:38.27  NormalTraffic   Normal_Min_Exec 19.437  >=  18.0
2012-03-15 12:40:48.896 AirPollution    AirPollution_AV_Exec    0.5195  >=  0.5
2012-03-15 12:41:02.821 AirPollution    AirPollution_AV_Exec    0.5181667   >=  0.5
2012-03-15 12:41:08.255 AirPollution    AirPollution_AV_Exec    0.5911429   >=  0.5
2012-03-15 12:41:14.631 AirPollution    AirPollution_AV_Exec    0.519125    >=  0.5
2012-03-15 12:41:21.034 AirPollution    AirPollution_AV_Exec    0.5743333   >=  0.5
2012-03-15 12:41:22.701 CallService CallService_AV_Through  18.0    <   25.5
2012-03-15 12:41:22.697 CallService CallService_MAX_Throu   18.0    <=  20.0
2012-03-15 12:41:22.82  DeviceConfiguration DeviceConfiguration_MAX_Throu   4.0 <   5.0
2012-03-15 12:41:22.675 IncidentAssessment  IncidentAssessment_MAX_Throu    2.0 <   3.0
2012-03-15 12:41:22.704 InformationService  InformationService_MAX_Throu    2.0 <   3.0
2012-03-15 12:41:27.441 AirPollution    AirPollution_AV_Exec    0.5187999999999999  >=  0.5
2012-03-15 12:41:34.882 AirPollution    AirPollution_AV_Exec    0.5639090999999999  >=  0.5
2012-03-15 12:41:40.26  AirPollution    AirPollution_AV_Exec    0.5181667   >=  0.5
2012-03-15 12:41:46.65  AirPollution    AirPollution_AV_Exec    0.55646155  >=  0.5
2012-03-15 12:41:52.046 AirPollution    AirPollution_AV_Exec    0.5179286   >=  0.5
2012-03-15 12:42:03.879 AirPollution    AirPollution_AV_Exec    0.5179375   >=  0.5
2012-03-15 12:42:15.675 AirPollution    AirPollution_AV_Exec    0.5177222   >=  0.5
2012-03-15 12:42:21.089 AirPollution    AirPollution_AV_Exec    0.5439474   >=  0.5
2012-03-15 12:42:26.442 AirPollution    AirPollution_AV_Exec    0.5175  >=  0.5
2012-03-15 12:42:49.728 NormalTraffic   NormalTrafficExec_Time  20.117  >=  20.03
2012-03-15 12:42:52.868 CallService CallService_AV_Through  13.5    <   25.5
2012-03-15 12:42:52.867 CallService CallService_MAX_Throu   18.0    <=  20.0
2012-03-15 12:42:52.864 CallService CallService_Throughput  9.0 <   10.0
2012-03-15 12:42:52.891 DeviceConfiguration DeviceConfiguration_AV_Through  3.0 <=  3.0
2012-03-15 12:42:52.89  DeviceConfiguration DeviceConfiguration_MAX_Throu   4.0 <   5.0
2012-03-15 12:42:52.889 DeviceConfiguration DeviceConfiguration_Min_Through 2.0 <   3.0
2012-03-15 12:42:52.888 DeviceConfiguration DeviceConfiguration_Throughput  2.0 <   4.0
2012-03-15 12:42:52.856 IncidentAssessment  IncidentAssessment_AV_Through   1.5 <=  1.5
2012-03-15 12:42:52.855 IncidentAssessment  IncidentAssessment_MAX_Throu    2.0 <   3.0
2012-03-15 12:42:52.852 IncidentAssessment  IncidentAssessment_Throughput   1.0 <   2.0
2012-03-15 12:42:52.876 InformationService  InformationService_AV_Through   1.5 <=  1.5
2012-03-15 12:42:52.874 InformationService  InformationService_MAX_Throu    2.0 <   3.0
2012-03-15 12:42:52.87  InformationService  InformationService_Throughput   1.0 <   2.0
2012-03-15 12:43:52.12  NormalTraffic   Normal_Max_Exec 43.156  >=  21.0
2012-03-15 12:43:52.106 NormalTraffic   NormalTrafficExec_Time  43.156  >=  20.03
2012-03-15 12:44:12.53  NormalTraffic   NormalTrafficExec_Time  20.365  >=  20.03
2012-03-15 12:44:12.557 NormalTraffic   Normal_Max_Exec 43.156  >=  21.0
2012-03-15 12:44:22.925 AccidentInformationProcess  AccidentInformationProcess_Min_Through  3.0 <   4.0
```

Figure 6.2: Violations of Second Execution.

```
2012-03-15 13:40:19.393 NoiseMeasure    NoiseMeasure_AV_Exec    1.045   >=  0.9
2012-03-15 13:40:19.409 NoiseMeasure    NoiseMeasure_Min_Exec   1.03    >=  1.0
2012-03-15 13:40:21.515 Assesement  Assesement_AV_Exec  2.059   >=  2.05
2012-03-15 13:40:21.515 Assesement  Assesement_Min_Exec 2.059   >=  2.03
2012-03-15 13:40:24.791 AirPollution    AirPollutionExecTime    1.046   >=  1.03
2012-03-15 13:40:24.791 AirPollution    AirPollution_AV_Exec    0.5305  >=  0.5
2012-03-15 13:40:26.912 Assesement  Assesement_AV_Exec  2.0515  >=  2.05
2012-03-15 13:40:26.943 Assesement  Assesement_Min_Exec 2.044   >=  2.03
2012-03-15 13:40:33.339 Assesement  Assesement_Min_Exec 2.044   >=  2.03
2012-03-15 13:40:37.551 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:40:37.551 AirPollution    AirPollution_AV_Exec    0.52275 >=  0.5
2012-03-15 13:41:41.792 CallService CallService_AV_Through  6.0 <   25.5
2012-03-15 13:41:41.792 CallService CallService_MAX_Throu   6.0 <=  20.0
2012-03-15 13:41:41.792 CallService CallService_Min_Through 6.0 <   9.0
2012-03-15 13:41:41.792 CallService CallService_Throughput  6.0 <   10.0
2012-03-15 13:41:53.211 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:42:05.052 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:42:16.924 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:42:39.45  NormalTraffic   NormalTrafficExec_Time  20.311  >=  20.03
2012-03-15 13:42:55.518 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:43:11.867 CallService CallService_AV_Through  12.0    <   25.5
2012-03-15 13:43:11.867 CallService CallService_MAX_Throu   18.0    <=  20.0
2012-03-15 13:43:11.867 CallService CallService_Min_Through 6.0 <   9.0
2012-03-15 13:43:19.183 NormalTraffic   NormalTrafficExec_Time  20.358  >=  20.03
2012-03-15 13:43:30.868 DeviceConfiguration DeviceConfigurationExecTime 1.03    >=  1.03
2012-03-15 13:43:35.314 AirPollution    AirPollutionExecTime    1.076   >=  1.03
2012-03-15 13:43:58.963 NormalTraffic   NormalTrafficExec_Time  20.39   >=  20.03
2012-03-15 13:44:15.125 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:44:35.421 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:44:35.436 AirPollution    AirPollution_AV_Exec    0.5061481600000001  >=  0.5
2012-03-15 13:44:38.681 NormalTraffic   NormalTrafficExec_Time  20.28   >=  20.03
2012-03-15 13:44:41.926 CallService CallService_AV_Through  18.666666666666668  <   25.5
2012-03-15 13:44:41.926 CallService CallService_Min_Through 6.0 <   9.0
2012-03-15 13:44:58.977 NormalTraffic   NormalTrafficExec_Time  20.233  >=  20.03
2012-03-15 13:45:15.091 AirPollution    AirPollution_AV_Exec    0.5072414   >=  0.5
2012-03-15 13:45:38.71  NormalTraffic   NormalTrafficExec_Time  20.327  >=  20.03
2012-03-15 13:45:54.794 AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:45:54.809 AirPollution    AirPollution_AV_Exec    0.50825806  >=  0.5
2012-03-15 13:46:12.016 Assesement  Assesement_Throughput   4.0 <   5.0
2012-03-15 13:46:12.0   CallService CallService_AV_Through  21.75   <   25.5
2012-03-15 13:46:12.0   CallService CallService_Min_Through 6.0 <   9.0
2012-03-15 13:46:12.016 NormalTraffic   Normal_Throughput   4.0 <   5.0
2012-03-15 13:46:11.844 DeviceConfiguration DeviceConfigurationExecTime 1.03    >=  1.03
2012-03-15 13:46:18.474 NormalTraffic   NormalTrafficExec_Time  20.42   >=  20.03
2012-03-15 13:46:34.636 AirPollution    AirPollution_AV_Exec    0.50915152  >=  0.5
2012-03-15 13:46:34.62  AirPollution    AirPollutionExecTime    1.03    >=  1.03
2012-03-15 13:46:58.27  NormalTraffic   NormalTrafficExec_Time  20.389  >=  20.03
2012-03-15 13:47:38.019 NormalTraffic   NormalTrafficExec_Time  20.342  >=  20.03
2012-03-15 13:47:42.044 CallService CallService_AV_Through  23.8    <   25.5
2012-03-15 13:47:42.044 CallService CallService_Min_Through 6.0 <   9.0
2012-03-15 13:48:17.83  NormalTraffic   NormalTrafficExec_Time  20.39   >=  20.03
```

Figure 6.3: Violations of Third Execution.

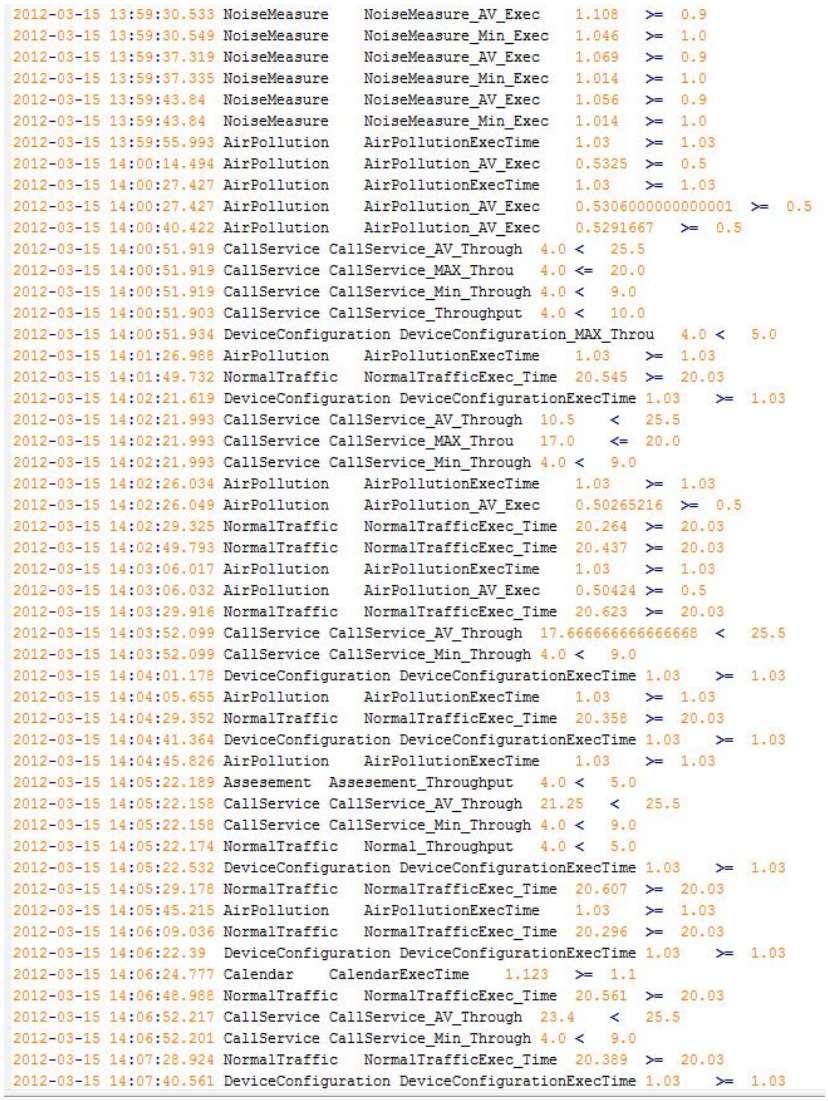


Figure 6.4: Violations of Fourth Execution.

6.2.4 Fourth Execution

The random integer used as input for the fourth execution is '7'. During this execution **'53'** violations were reported and the first '51' are depicted in Figure 6.4.

During the fourth execution, twenty times an accident did not occur, two times we had high noise pollution , twenty-four times high air pollution and forty-three times heavy traffic. According to Table 6.2, none of these properties is violated.

```
2012-03-16 15:56:30.431 AccidentInformationProcess  AccidentExec_Tim    2.141   >=  2.1
2012-03-16 15:56:30.491 AccidentInformationProcess  AccidentInformationProcess_Min_Exec 2.141   >=  2.1
2012-03-16 15:56:31.72  Calendar    Calendar_Min_Exec   1.052   >=  1.05
2012-03-16 15:56:34.629 Assesement  AssesementExecTime  2.152   >=  2.1
2012-03-16 15:56:34.642 Assesement  Assesement_AV_Exec  2.135   >=  2.05
2012-03-16 15:56:34.644 Assesement  Assesement_Min_Exec 2.152   >=  2.03
2012-03-16 15:56:37.99  AirPollution    AirPollution_AV_Exec    0.613   >=  0.5
2012-03-16 15:56:40.132 Assesement  Assesement_AV_Exec  2.076   >=  2.05
2012-03-16 15:56:46.61  Assesement  Assesement_AV_Exec  2.0566667   >=  2.05
2012-03-16 15:56:49.9   AirPollution    AirPollution_AV_Exec    0.56625 >=  0.5
2012-03-16 15:57:02.921 AirPollution    AirPollution_AV_Exec    0.5535  >=  0.5
2012-03-16 15:57:54.342 CallService CallService_AV_Through  6.0 <   25.5
2012-03-16 15:57:54.341 CallService CallService_MAX_Throu   6.0 <=  20.0
2012-03-16 15:57:54.34  CallService CallService_Min_Through 6.0 <   9.0
2012-03-16 15:57:54.338 CallService CallService_Throughput  6.0 <   10.0
2012-03-16 15:58:49.687 DeviceConfiguration DeviceConfigurationExecTime 1.031   >=  1.03
2012-03-16 15:58:49.695 DeviceConfiguration DeviceConfiguration_AV_Exec 1.025   >=  1.0
2012-03-16 15:58:49.732 DeviceConfiguration DeviceConfiguration_Min_Exec    1.031   >=  1.0
2012-03-16 15:59:17.548 NormalTraffic   NormalTrafficExec_Time  20.202  >=  20.03
2012-03-16 15:59:24.4   CallService CallService_AV_Through  11.0    <   25.5
2012-03-16 15:59:24.398 CallService CallService_MAX_Throu   16.0    <=  20.0
2012-03-16 15:59:24.397 CallService CallService_Min_Through 6.0 <   9.0
2012-03-16 15:59:24.418 DeviceConfiguration DeviceConfiguration_MAX_Throu   4.0 <   5.0
2012-03-16 15:59:24.405 InformationService  InformationService_MAX_Throu    2.0 <   3.0
2012-03-16 15:59:56.984 NormalTraffic   NormalTrafficExec_Time  20.202  >=  20.03
2012-03-16 16:00:17.153 NormalTraffic   NormalTrafficExec_Time  20.127  >=  20.03
2012-03-16 16:00:54.483 Assesement  Assesement_Throughput   4.0 <   5.0
2012-03-16 16:00:54.451 CallService CallService_AV_Through  18.0    <   25.5
2012-03-16 16:00:54.449 CallService CallService_Min_Through 6.0 <   9.0
2012-03-16 16:00:54.479 NormalTraffic   Normal_Throughput   4.0 <   5.0
2012-03-16 16:00:56.559 NormalTraffic   NormalTrafficExec_Time  20.157  >=  20.03
2012-03-16 16:01:36.056 NormalTraffic   NormalTrafficExec_Time  20.157  >=  20.03
2012-03-16 16:02:15.406 NormalTraffic   NormalTrafficExec_Time  20.137  >=  20.03
2012-03-16 16:02:24.502 CallService CallService_AV_Through  21.25   <   25.5
2012-03-16 16:02:24.5   CallService CallService_Min_Through 6.0 <   9.0
2012-03-16 16:02:55.004 NormalTraffic   NormalTrafficExec_Time  20.355  >=  20.03
2012-03-16 16:03:34.619 NormalTraffic   NormalTrafficExec_Time  20.311  >=  20.03
2012-03-16 16:03:54.555 CallService CallService_AV_Through  23.6    <   25.5
2012-03-16 16:03:54.553 CallService CallService_Min_Through 6.0 <   9.0
2012-03-16 16:04:14.128 NormalTraffic   NormalTrafficExec_Time  20.239  >=  20.03
```

Figure 6.5: Violations of Fifth Execution.

6.2.5 Fifth Execution

The random integer used as input for the execution is '2'. During this execution **'41'** violations were reported (Fig. 6.5).

During the fifth execution, twenty-one times an accident did not occur, one time we had high noise pollution, twenty-three times high air pollution and thirty-nine times heavy traffic. According to Table 6.2, none of these properties is violated.

6.2.6 Aggregated Results

Tables 6.3 and 6.4, summarize the aforementioned results for each process and execution. When a number of an execution is not depicted at the table, this means that at the corresponding execution no violations occurred .

Process		Execution Time				Throughput			
		Min (sec)	Max (sec)	Average (sec)	Class (sec)	Min	Max	Average	Class
AccidentInformationProcess	2					1 violation (3.0)		1 violation (6.333333333333333)	1 violation (3.0)
	3					1 violation (3.0)			1 violation (3.0)
	5	1 violation (2.141)			1 violation (2.141)				
AirPollution	1			10 violations (0.5525, 0.7103333000000001 0.53675 ,0.5525, 0.52625, 0.5235,0.5211667 ,0.5181539000000001 0.5179642999999999, 0.50072415)					
	2			15 violations (0.523, 0.5195, 0.5181667, 0.5911429, 0.519125, 0.5743333, 0.5187999999999999, 0.5639090999999999, 0.5181667, 0.5564155, 0.5179286, 0.5179375, 0.517722)2, 0.5439474, 0.5175)		1 violation (3.0)		1 violation (6.166666666666667)	1 violation (3.0)
	3			6 violations (0.5305, 0.52275, 0.5061481600000001, 0.5072414, 0.50825806, 0.50915152)	3 violations (1.046, 1.03, 1.076)				
	4			5 violations (0.5325, 0.5306000000000001, 0.5291667, 0.50265216, 0.50424)	1 violation (1.03)				
	5			3 violations (0.613, 0.56625, 0.5535)					

Process		Execution Time				Throughput			
		Min (sec)	Max (sec)	Average (sec)	Class (sec)	Min	Max	Average	Class
Assesement	1				1 violation (2.135)				1 violation (4.0)
	2					1 violation (3.0)	1 violation (4.0)		1 violation (3.0)
	3	1 violation (2.059, 2.044)		2 violations (2.059, 2.0515)					1 violation (4.0)
	4						1 violation (4.0)		1 violation (4.0)
	5	1 violation (2.152)		3 violations (2.135,2.076, 2.0566667)	1 violation (2.152)		1 violation (4.0)		
CallService	1	1 violation (1.984)	1 violation (1.984)	11 violations (1.975, 1.515, 1.3463334 ,1.262, 1.211, 1.1771666, 1.1532858000000001, 1.13475 1.1206666, 1.1094000000000002 1.1002726999999999)	1 violation (1.984)	1 violation (8.0)	2 violations (8.0, 16.0)	6 violations (8.0, 12.0 , 18.666666666666668, 21.75, 24.0 ,24.666666666666668)	1 violation (8.0)
	2						1 violations (18.0,)	6 violations (18.0, 13.5, 16.666666666666668, 19.5, 22.0, 23.5)	1 violation (9.0)
	3					1 violation (6.0)	2 violations (6.0, 18.0)	6 violations (6.0, 12.0, 18.666666666666668, 21.75, 23.8, 24.333333333333332)	1 violation (6.0)
	4					1 violation (4.0)	2 violations (4.0, 17.0)	5 violations (4.0, 10.5, 17.666666666666668, 21.25, 23.4)	1 violation (4.0)
	5					1 violation (6.0)	2 violations (6.0, 16)	5 violations (6.0, 11,18,21.25,23.6)	1 violation (6.0)
Calendar	2						1 violation (9.75)	2 violations (9.6, 9.5)	
	4				1 violation (1.123)				
	5	1 violation (1.052)							

Table 6.3: Aggregated Results.

Process		Execution Time				Throughput			
		Min (sec)	Max (sec)	Average (sec)	Class (sec)	Min	Max	Average	Class
DeviceConfiguration	1	1 violation (1.076)	1 violation (1.076)	1 violation (1.069)	1 violation (1.076)		1 violation (4.0)		
	2					1 violation (2.0)	1 violation (4.0)	1 violation (3.0)	1 violation (2.0)
	3				1 violation (1.03)				
	4				1 violation (1.03)		1 violation (4.0)		
	5	1 violation (1.031)		1 violation (1.025)	1 violation (1.031)		1 violation (4.0)		
IncidentAssessment	2						1 violation (2.0)	1 violation (1.5)	1 violation (1.0)
InformationService	1						1 violation (2.0)		
	2						1 violation (2.0)	1 violations (1.5)	1 violation (1.0)
	5						1 violation (2.0		
NoiseMeasure	1	1 violation (1.057)		1 violation (1.123)					
	2	1 violations (1.017)		1 violation (1.039)			1 violation (3.0)		1 violation (3.0)
	3	1 violation (1.045)		1 violation (1.03)					
	4	2 violations (1.046, .014)		3 violations (1.108, 1.069,1,056)					

Process		Execution Time				Throughput			
		Min (sec)	Max (sec)	Average (sec)	Class (sec)	Min	Max	Average	Class
Normal Traffic	1				10 violations (20.216,20.213 20.15, 20.089 20.181,20.118 20.229,20.225 20.212,20.214)				1 violations (4.0)
	2	2 violations (20.252,19.437	1 violation (43.156)	1 violation (20.261)	12 violations (20.252, 20.117, 43.156, 20.365, 20.106, 20.222, 35.271, 20.112, 20.238, 20.259, 20.25, 20.25)		1 violation (3.0)	1 violation (6.166666666 666667)	2 violations (3.0, 4.0)
	3				12 violations (20.311, 20.358, 20.39, 20.28, 20.233, 20.327, 20.233, 20.42, 20.389, 20.342, 20.39, 20.467)				1 violation (4.0)
	4				8 violations (20.545, 20.264, 20.437, 20.623, 20.358, 20.607, 20.296, 20.561)				1 violation (4.0)
	5				7 violations (20.202, 20.127, 20.157, 20.355, 20.311, 20.239, 20.348)				1 violation (4.0)

Table 6.4: Aggregated Results.

6.3 Check Availability

According to the aforementioned results availability was never violated. In order to test if our framework is able to catch violations regarding to availability we created a program that randomly deploys and un-deploys services. The program runs for ten minutes and Fig. 6.6 depicts the deployment and un-deployment of services. During the execution of the program, the processes were un-deployed and re-deployed with the following order:

1. AirPollution.
2. InformationService.
3. NormalTraffic.
4. Assesement.
5. Assesement.
6. NoiseMeasure.
7. InformationService.
8. CallService
9. AccidentInformation
10. AirPollution.
11. Assememt.
12. Calendar.
13. IncidentAssessment.
14. Calendar.
15. Assememt.
16. NoiseMeasure.
17. CallService
18. CallService
19. Assememt.
20. DeviceConfiguration.
21. NormalTraffic.
22. CallService
23. Assememt.
24. InformationService.
25. NoiseMeasure.
26. IncidentAssessment.
27. AccidentInformation
28. AccidentInformation
29. CallService
30. InformationService.
31. InformationService.
32. DeviceConfiguration.
33. NoiseMeasure.
34. NormalTraffic.
35. NormalTraffic.
36. CallService
37. Calendar.
38. AirPollution.
39. Calendar.
40. Calendar.
41. Calendar.
42. InformationService.
43. NormalTraffic.
44. InformationService.
45. CallService
46. CallService
47. IncidentAssessment.
48. NoiseMeasure.
49. CallService
50. NormalTraffic.
51. AirPollution.

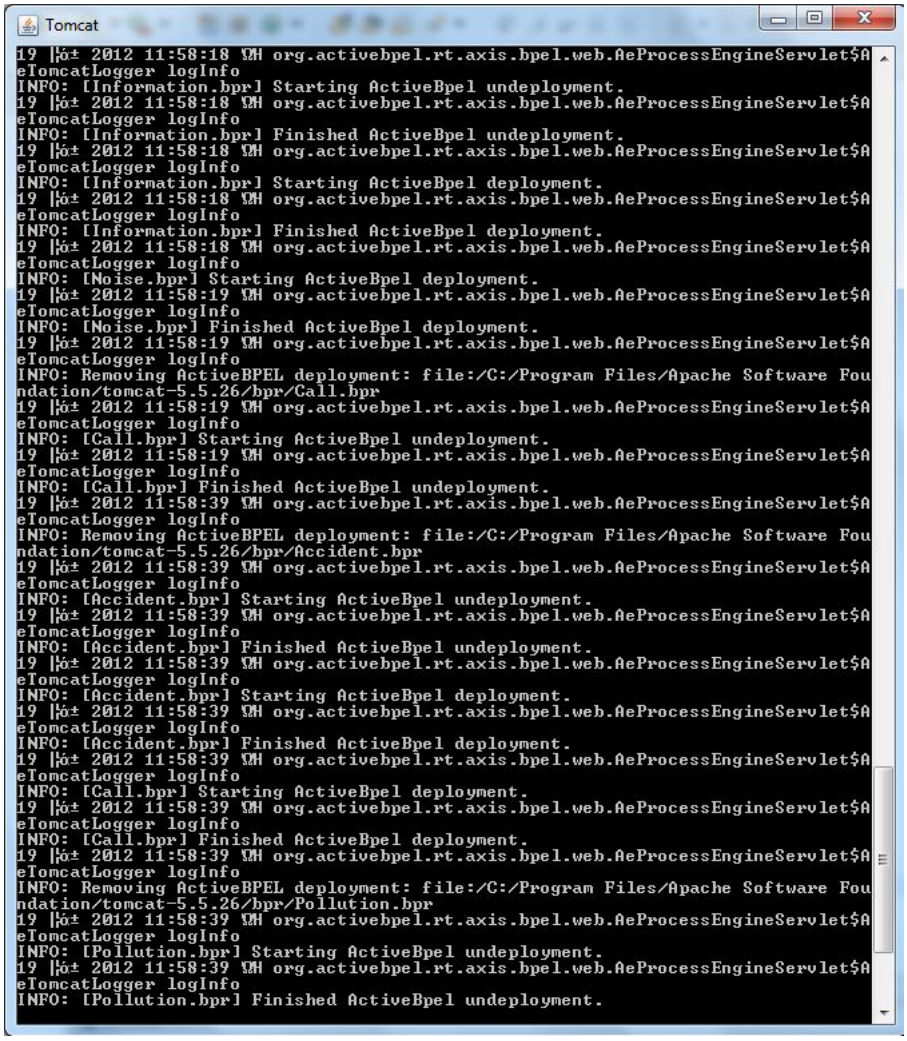


Figure 6.6: Deployment-Un-deployment of Services (Tomcat Console).

52. IncidentAssessment.
53. Calendar.
54. NoiseMeasure.
55. IncidentAssessment.
56. Assememt.
57. Calendar.
58. AirPollution.

During the execution '197' violations were reported and the first '51' are depicted in Figure 6.7.

```
2012-03-19 11:58:00.892 Assesement  Assesement_Availability 0.7 <=  0.99
2012-03-19 11:58:00.897 Assesement  Assesement_Aver_Avail   0.7 <=  0.99
2012-03-19 11:58:00.896 Assesement  Assesement_Max_Avail    0.7 <   0.99
2012-03-19 11:58:00.894 Assesement  Assesement_Min_Avail    0.7 <   0.99
2012-03-19 11:58:00.846 InformationService  InformationService_Availability 0.6 <=  0.99
2012-03-19 11:58:00.852 InformationService  InformationService_Aver_Avail   0.6 <=  0.99
2012-03-19 11:58:00.85  InformationService  InformationService_Max_Avail    0.6 <=  0.99
2012-03-19 11:58:00.848 InformationService  InformationService_Min_Avail    0.6 <=  0.99
2012-03-19 11:58:00.874 NoiseMeasure    NoiseMeasure_Availability   0.9 <=  0.99
2012-03-19 11:58:00.879 NoiseMeasure    NoiseMeasure_Aver_Avail 0.9 <=  0.99
2012-03-19 11:58:00.878 NoiseMeasure    NoiseMeasure_Max_Avail  0.9 <=  0.99
2012-03-19 11:58:00.876 NoiseMeasure    NoiseMeasure_Min_Avail  0.9 <=  0.99
2012-03-19 11:59:01.324 AirPollution    AirPollution_Availability   0.7 <=  0.99
2012-03-19 11:59:01.327 AirPollution    AirPollution_Aver_Avail 0.85    <=  0.99
2012-03-19 11:59:01.325 AirPollution    AirPollution_Min_Avail  0.7 <=  0.99
2012-03-19 11:59:01.4   Assesement  Assesement_Aver_Avail   0.85    <=  0.99
2012-03-19 11:59:01.399 Assesement  Assesement_Min_Avail    0.7 <   0.99
2012-03-19 11:59:01.367 Calendar    Calendar_Availability   0.9 <=  0.99
2012-03-19 11:59:01.37  Calendar    Calendar_Aver_Avail 0.95    <=  0.99
2012-03-19 11:59:01.368 Calendar    Calendar_Min_Avail  0.9 <=  0.99
2012-03-19 11:59:01.34  CallService CallService_Availability    0.7 <=  0.99
2012-03-19 11:59:01.346 CallService CallService_Aver_Avail  0.85    <=  0.99
2012-03-19 11:59:01.342 CallService CallService_Min_Avail   0.7 <=  0.99
2012-03-19 11:59:01.364 InformationService  InformationService_Aver_Avail   0.8 <=  0.99
2012-03-19 11:59:01.362 InformationService  InformationService_Min_Avail    0.6 <=  0.99
2012-03-19 11:59:01.379 NoiseMeasure    NoiseMeasure_Availability   0.7 <=  0.99
2012-03-19 11:59:01.382 NoiseMeasure    NoiseMeasure_Aver_Avail 0.8 <=  0.99
2012-03-19 11:59:01.381 NoiseMeasure    NoiseMeasure_Max_Avail  0.9 <=  0.99
2012-03-19 11:59:01.38  NoiseMeasure    NoiseMeasure_Min_Avail  0.7 <=  0.99
2012-03-19 12:00:01.829 AirPollution    AirPollution_Aver_Avail 0.9 <=  0.99
2012-03-19 12:00:01.825 AirPollution    AirPollution_Min_Avail  0.7 <=  0.99
2012-03-19 12:00:01.899 Assesement  Assesement_Aver_Avail   0.9 <=  0.99
2012-03-19 12:00:01.897 Assesement  Assesement_Min_Avail    0.7 <   0.99
2012-03-19 12:00:01.872 Calendar    Calendar_Availability   0.4 <=  0.99
2012-03-19 12:00:01.874 Calendar    Calendar_Aver_Avail 0.7666666666666666  <=  0.99
2012-03-19 12:00:01.873 Calendar    Calendar_Min_Avail  0.4 <=  0.99
2012-03-19 12:00:01.849 CallService CallService_Availability    0.9 <=  0.99
2012-03-19 12:00:01.853 CallService CallService_Aver_Avail  0.8666666666666667  <=  0.99
2012-03-19 12:00:01.851 CallService CallService_Min_Avail   0.7 <=  0.99
2012-03-19 12:00:01.869 InformationService  InformationService_Aver_Avail   0.8666666666666667  <=  0.99
2012-03-19 12:00:01.867 InformationService  InformationService_Min_Avail    0.6 <=  0.99
2012-03-19 12:00:01.891 NoiseMeasure    NoiseMeasure_Availability   0.7 <=  0.99
2012-03-19 12:00:01.893 NoiseMeasure    NoiseMeasure_Aver_Avail 0.7666666666666666  <=  0.99
2012-03-19 12:00:01.892 NoiseMeasure    NoiseMeasure_Max_Avail  0.9 <=  0.99
2012-03-19 12:00:01.892 NoiseMeasure    NoiseMeasure_Min_Avail  0.7 <=  0.99
2012-03-19 12:01:02.253 AirPollution    AirPollution_Aver_Avail 0.925   <=  0.99
2012-03-19 12:01:02.251 AirPollution    AirPollution_Min_Avail  0.7 <=  0.99
2012-03-19 12:01:02.271 CallService CallService_Availability    0.4 <=  0.99
2012-03-19 12:01:02.274 CallService CallService_Aver_Avail  0.75    <=  0.99
2012-03-19 12:01:02.272 CallService CallService_Min_Avail   0.4 <=  0.99
2012-03-19 12:01:02.278 DeviceConfiguration DeviceConfiguration_Availability    0.7 <=  0.99
```

Figure 6.7: Reported violations for Availability.

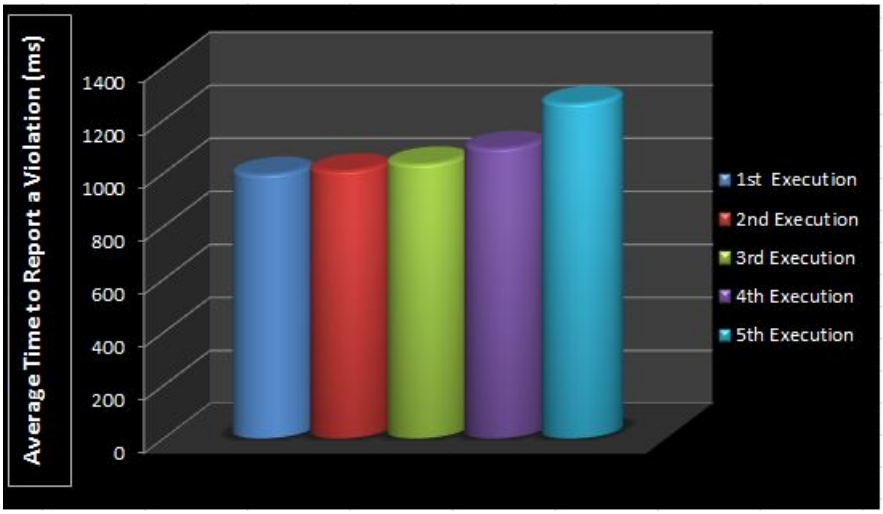


Figure 6.8: Average Time Needed to Report a Violation.

6.4 Conclusion

The objective of our work was not only to report the monitoring properties and catch the occurred violations, but also to report these violations on time. So, we measure the delay in reporting a violation. For each of the above 5 executions, we evaluate the average time in milliseconds to report a violation. In Figure 6.8, we provide a diagram with the results of each execution. As seen, our monitoring framework is able to report the occurred violations on time. The average time needed to report a violation is one second

Chapter 7

Related Work

7.1 Approaches to Monitoring of Service-Based Systems

The need to monitor SOAs at run-time has inspired a large number of research projects, both academic and industrial. The differences between these research proposals are manifold, and quite evident after an accurate analysis. This has led to an unfortunate situation in which the term monitoring is commonly used, but with many possible interpretations. Although their main goal discovering potential critical problems in an executing system-remains the same, there are differences that concern important aspects, such as the goals of monitoring, the stakeholders who might be interested in them, the potential problems one might try to detect, etc.

In this section we survey the state-of-the-art works in monitoring of service-based applications. We describe a number of different approaches and at section 7.2 a comparison of them is provided.

7.1.1 Smart monitors for composed services

In [9] the authors proposed an approach towards monitoring of service compositions specified as BPEL processes against contracts expressed as assertions on service. In particular, the assertion defines the pre- and post-conditions on service invocations, and characterizes functional expectations of the correct service behavior. The authors propose the process annotations as a way of specifying the contract assertions. These annotations are then automatically translated into the corresponding code extensions that interact

with the dedicated monitor components in order to perform the assertion checking. In their proposal these monitors are web services themselves, which receive the instructions from the transformed process, perform the analysis and return the result to the executed process.

In order to implement the monitoring functionality, the authors present two complementary approaches. The first one relies on the C# object-oriented language for specifying and implementing monitor specifications. In this approach, the assertions are expressed completely in this language, thus providing very expressive means for the specification. The corresponding monitor is then automatically generated in this language and deployed as a Web service. The approach is very flexible and expressive; however, it requires the process designer to work at a very low programming level. The other approach relies on the use of CLIX, a rich and expressive XML-based assertion language, based on first-order logic, which supports, for instance, quantifiers. The implementation of the monitor in this case relies on wrapping the corresponding assertion processor as a Web service. The approach is less flexible, but benefits from high-level and declarative assertion specification language.

7.1.2 Dynamo

In [10, 11, 8], Baresi and Guinea extend and elaborate the ideas presented in previous approach [9].The main challenges for the new approach are:

- to enable separation of concerns in the design of monitoring specification so that new monitoring rules are defined separately from the process itself;
- to come up with a well-defined and expressive language for the monitoring specifications;
- to provide means for monitoring non-functional (QoS) properties of the services;
- to support the collection of the information from external sources;
- to provide means for managing monitoring process;
- to devise and implement the corresponding architecture.

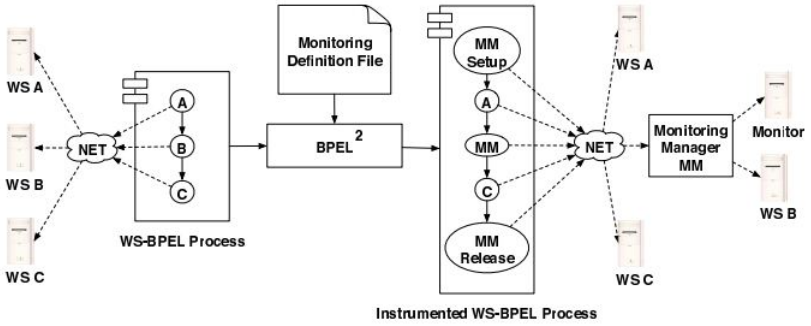


Figure 7.1: Dynamo Monitoring Approach

The authors propose an extended notation for specifying monitoring rules. Monitoring rules are defined explicitly and externally in a well-defined structured file. This separation allows different sets of rules to be associated with same process. Monitoring rules abstract Web services into suitable UML classes, and use this abstraction to specify constraints on execution. The rules define the location, where the evaluation should be performed, and its type. For precondition and postcondition, location indicates the BPEL invoke activity, to which the rule is applied; for an invariant it indicates the BPEL scope; and in case of assertion it indicates any point of WS-BPEL process before the assertion should hold.

Monitoring expressions are specified in WS-CoL (Web Service Constraint Language [10]) , a special-purpose assertion specification language that borrows its roots from JML (Java Modeling Language [36]), and extends it with constructs to gather data from external sources (i.e., to interact with external data collectors). The language enables specifying expressions over the process variables and supports a set of built-in functions, logical and mathematical operators, and quantification.

Besides constraining the execution, monitoring rules provide parameters to govern the degree of run-time checking. After weaving selected rules into the process at deployment time, the user can set the amount of monitoring at run-time by means of these parameters. The monitoring rules are deployed together with the process through a weaving procedure, i.e., parsing monitoring rules and adding specific WS-BPEL activities to process in order to achieve dynamic monitoring. At run-time the modified process interacts with the proxy service, namely rule manager, which is responsible for processing the monitoring management instructions, processing monitor configuration, obtaining information from external

data sources, evaluating monitoring expressions, and interacting with the actual services (instead of the original process). If some constraints are not met, monitoring manager is responsible to inform BPEL process. Fig.7.1 illustrates this monitoring approach.

In [8] the authors extend this work for what concerns the kind of properties the approach can monitor. The extended specification language, namely TimedWSCoL now allows for specifying temporal properties over the events that occur during the process execution. In particular, the authors present classical linear temporal operators (always, sometime, until) and specific operators to express a property over restricted time window (within, between, count). The monitoring of these temporal properties is asynchronous, and performed by a new dedicated component called WSCoL Analyzer. This analyzer is executed in parallel with the execution of a process, and receives the relevant event through a dedicated publish/subscribe mechanism. The automata-based algorithms are used to devise the monitor for a dedicated temporal property.

In [11] the authors show how this approach may be integrated with the WS-Policy framework [6].

WS-Policy is emerging as the standard way to describe the properties that characterize a Web service. By means of this specification, the functional description of a service can be tied to a set of assertions that describe how the Web service should work in terms of aspects like security, transactionality, and reliable messaging. These assertions can be used to express both functional and non-functional aspects. Policies can be defined by several actors and during different phases of the Web service life-cycle. Besides implementing the application, service developers also specify the properties that must hold during the execution regardless of the platform on which the services will be deployed (service policies). On the other hand, service providers specify the features supported by the application servers on which services are deployed (server policies). The intersection of service and server policies results in supported policies, which define the properties of the services deployed on a specific platform. Finally, Web service users state the features that should be supported by the services they want to invoke (requested policies). By combining requested policies and supported policies, we obtain the so called *effective policies*. Effective policies represent the set of assertions that specify the properties of a Web service deployed on a particular server and invoked by a specific user.

In [11] the author's work concentrates on the effective policies. Once effective policies are derived, services should be monitored at run-time to guarantee that they offer the service levels stated by their associated policies.

7.1.3 Requirements monitoring based on event calculus

In [39, 38, 46], the authors approach the problem of monitoring service-based systems for conformance to a set of behavioral properties and assumptions. Behavioral properties are automatically extracted from the specification of the composition process of the SBS system and assumptions are additional requirements about the behavior of agents interacting with the system, or the individual services of it. In particular, the works address the ability to represent and monitor complex and expressive properties that deal with events, states, timing constraints and relations. Furthermore, there is a need to refer not only to functional but also to non-functional characteristic of the system, as dictated by the necessity of the run-time compliance checking between the actual behavior and the service-level agreement specifications [39].

The service level agreements that can be monitored by the framework (Fig.:7.2) are expressed using an extension of WS-Agreement [3]. This extension supports the description of: (a) the operational context of an agreement, (b) the policy for monitoring an agreement, and (c) the functional and quality requirements for the service which is regulated by the agreement and need to be monitored (i.e., the guarantee terms in the terminology of WS-Agreement). The extensions of WS-Agreement (a) and (b) have been directly integrated in the XML schema that defines this language. To support the specification of (c), the authors developed a new language in which service guarantee terms are specified in terms of: (i) events which signify the invocation of operations of a service by the composition process of an SBS system and returns from these executions, (ii) events which signify calls of operations of the composition process of an SBS system by external services and returns from these executions, and (iii) the effects that events of either of the above kinds have on the state of an SBS system or the services that it deploys (e.g., change of the values of system variables). This language has been defined by a separate XML schema and is called EC-Assertion. EC-Assertion iis based on Event Calculus (EC) [45] which is a first order temporal logic language. Specifications of service guarantee terms in EC-

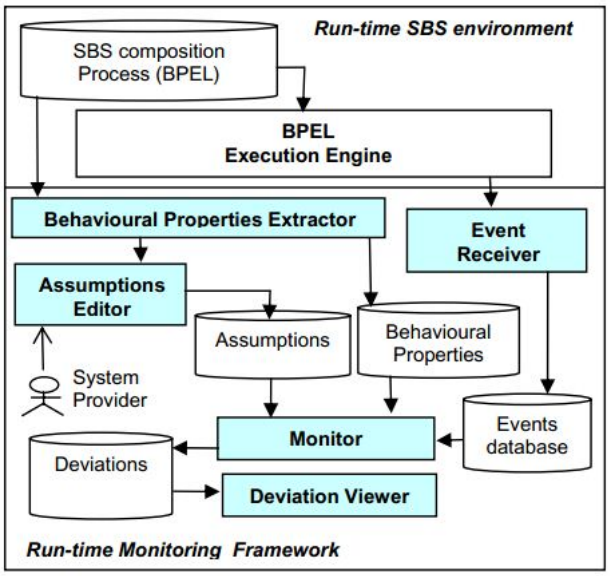


Figure 7.2: Requirements Monitoring Approach

Assertion can be developed independently of WS-Agreement and subsequently referred to by it.

Monitoring is performed in parallel with the normal operation of an SBS system without interrupting it. This is possible by intercepting events which are exchanged between the composition process of an SBS system and its services and the effects of these events on the state of the composition process of the system. This approach makes run-time monitoring non intrusive as: (a) it does not affect the performance of SBS systems, and (b) it does not require the instrumentation of the code of the composition process of SBS systems or their services to generate the events which are required for monitoring. Furthermore, the framework can monitor three different types of deviations from service guarantee terms including: (a) violations of terms by the recorded behavior of a system, (b) violations of service guarantee terms by the expected behavior of the system and (c) cases of unjustified system behavior that may arise when a system acts incorrectly due to incorrect information about its state.

The monitoring framework was implemented as a toolkit for monitoring service compositions specified in BPEL. The logs generated by the process engine were used to identify the events and update the corresponding formula templates in the monitors. In order to

evaluate and validate the presented approach, the authors set up a comprehensive benchmark with many test and generated events based on a simple case study parametrized by the frequency of events and the scale of the involved components.

7.1.4 Planning and monitoring execution with business assertions

In [35] Lazovik et al. apply monitoring to a framework, where the user requests are used to dynamically customize and execute standard business processes. Such customization aims at satisfying the user constraints and requirements to the execution of a standard business process, parametric to the set of available concrete services participating in the process. These services, however, as well as their composition, should satisfy certain domain business rules, referred to in the framework as assertions.

Three kinds of assertions are supported depending on their operational context and complexity: simple assertions, where simple reachability conditions are checked; preservation assertions, where maintenance of some condition needs to be satisfied throughout a path comprising a set of states traversed by the process during execution time; and business entity assertions, where the evolution sequence of a particular variable is monitored for correctness. Assertions are specified in the assertion language XSAL (Xml Service Assertion Language).

In this approach the authors propose an architecture, where the planning-based adaptation of the business process is interleaved with the execution and monitoring of the process and the corresponding assertions. The adaptation requests are specified in a XSRL (Xml Service Request Language) query language that defined functional constraints and preferences of the user.

This framework is based on reactive monitoring. In particular, designers can define three kinds of properties: (1) goals that must be true before transitioning to the next state (2) goals that must be true for the entire process execution, and (3) goals that must be true for the process execution and evolution sequence. The XSRL language also allows for the definition of constraints as boolean combinations of linear inequalities and boolean propositions. It provides sequencing operators such as achieve-all, before and then, prefer goal x to goal y, and then. It also defines a number of operators that can be used on the propositions themselves, defining how these propositions should be satisfied such as vital

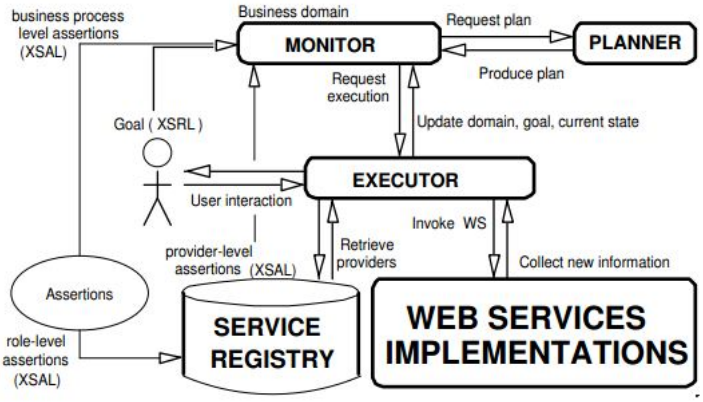


Figure 7.3: Planning and monitoring framework

and optional. The delivery platform continuously loops between execution and planning. In particular, the latter activity is achieved by taking into account the context and the properties specified for the state-transition system. This makes it possible to discover, each time it is undertaken, whether a property has been violated by the previously executed step, or if execution is proceeding correctly. In case a violation of the plan or an assertion is detected, the platform tries to dynamically modify the plan taking into account new situation and assertions.

This approach is presented in Fig. 7.3

7.1.5 Cremona

In [37] Ludwig et al. propose an architecture and implementation for creation, management and monitoring service-level agreements represented as WS-Agreement documents. Cremona is a proposal from IBM and stands for "Creation and Monitoring of WS-Agreements".

Regarding the monitoring problem, the Cremona framework provides an Agreement Provider component, whose structure incorporates, among other things, a Status Monitor. This component is specific to the system providing the service. By consulting the resources available on the system and the terms of an agreement, it helps decide whether a negotiation proposal should be accepted or refused. Once an agreement has been accepted by both parties (the client and the provider), its validity is checked at run-time by a Compliance Monitor, a sophisticated system-specific component that can check for

violations as they occur, predict violations that still have to occur, and take corrective actions. Since both monitoring components are system dependent, designers are guaranteed great flexibility in terms of the properties they can check.

7.1.6 Colombo

In [21] the authors propose a platform for developing, deploying, and executing service-oriented applications and system that incorporates the tools and facilities for checking, monitoring, and enforcing service requirements expressed in WS-Policy notations. Apart from checking the compliance of policies at deployment-time, it is necessary to verify them at run-time, when, e.g., service invocations calls/bindings take place or messages are sent/received.

The Colombo platform comes with the module that manages the policy assertions. Apart from evaluating the assertions attached to particular service-related entity, the framework provides means for policy enforcement, e.g., it may approve the delivery of a message, reject the delivery, or defer further processing.

Colombo is a lightweight middleware for service-oriented architectures proposed by IBM. It advocates that an optimized and native run-time environment, which does not build upon previously existing application servers, can provide greater performance, and guarantee simplified models for development and deployment. It supports the entire web service stack and, in particular, orchestrated collaborations defined using BPEL.

7.1.7 Glassfish

GlassFish [31] is an open-source community implementation of a server for Java EE 5 applications. Regarding monitoring of deployed services, GlassFish provides a number of specific tools. Using technologies such as 'J2EE Management' and 'Java Management Extensions', GlassFish makes it possible to access information on resources and properties that are tied to the web services to be monitored. This information is given in the form of operational statistics (and in graphical form as well). The nature of the monitored aspects depends on the level of monitoring chosen for a given service.

There are three possible levels:

- ‘ **Low**’: which monitors response times, throughput, and the total number of requests and faults;
- ‘ **Medium**’: which adds message tracing under the form of content visualization;
- ‘ **Off**’: in which no data is collected.

Captured information can also be automatically aggregated to obtain minimum response times, maximum response times, average response times, etc.

Analysis of the monitored data could be achieved either manually, or automatically, possibly in conjunction with a more sophisticated monitoring approach, such as Dynamo.

7.1.8 Query-based business process monitoring

In [12] Beeri et al. propose an approach to the monitoring of business processes specified in BPEL. While the goal of the monitoring approach is similar to many other proposed approaches, the specific focus of the approach is different. In particular, the authors try to address the following monitoring design and implementation issues: the monitor specification should target the same level of abstraction as the original process; the monitoring activity should take into account the specific features of the underlying process models; the monitors should be deployed and executed in the same environment as the original process, without putting additional requirements on that environment.

For these purposes the Business Process Monitoring (BP-Mon) system is presented (Fig. 7.4). In order to specify the monitoring properties, a novel query language for monitoring business processes is proposed, that allows users to visually define monitoring tasks and associated reports, using a simple intuitive interface, similar to those used for designing BPEL processes. BP-Mon queries the actual execution flow of a live and running BPEL process and therefore allows to monitor processes at run time. Queries consist of two main ingredients: execution patterns that should be matched against the actual execution traces, and report specification generated from these matches. The patterns represent the composite events as partial control flow specifications, where the elements specify the activities that should appear in the critical execution. Additionally, the query contains the definition of the time window (period and interval), in which the query should be evaluated, and the condition to restrict the set of matched executions.

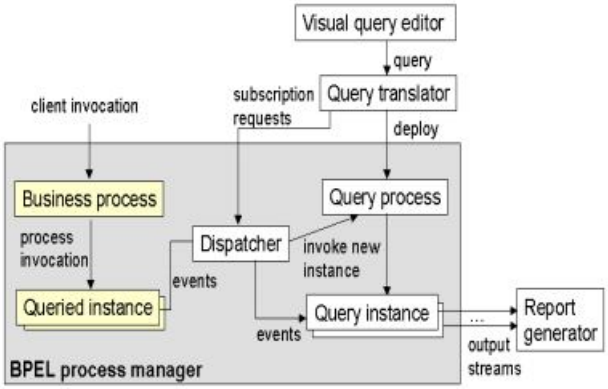


Figure 7.4: BP-Mon Architecture

When the query is matched, the report is populated and emitted. The report represents a parametric XML pattern, which is instantiated when the pattern is matched. Two reporting modes are provided: a local one, where an individual report is issued for each process instance and a global one that considers all the instances.

To perform the run-time analysis, a specific pattern matching algorithm is proposed. The algorithm tries to greedily simulate the pattern to match events as early as possibly and backtracking on failure. A report is issued as soon as a match for the pattern is identified.

The system is implemented as follows. A BP-Mon query is compiled into a BPEL process specification, whose instances perform the monitoring task, which is translated into an executable code to be run on same BPEL engine as the monitored business process. An additional component, dispatcher, is used to listen to the events on the process activities, and forward them to the query process instance. An important feature of the approach is that it does not target a particular monitoring goal. Indeed, the reports provide just the required values, and therefore may be used for various purposes regarding BPEL processes.

7.2 Comparing monitoring approaches

A summary of all the aforementioned approaches is presented in Table 7.1. The approaches are compared according to intrusiveness and timeliness. The timeliness of a monitoring system presents the ability of the framework to signal violations of the

Approach Name	Intrusiveness	Collaboration Paradigm	Timeliness	Type of properties	
				Kind	Scope
Dynamo [10, 11,8]	No	BPEL-based orchestrations	Signal information of interest as soon as they occur	Functional Non- Functional	Instance
Smart monitors [9]	Yes	BPEL-based orchestrations	Signal information of interest as soon as they occur	Functional	Instance
Requirements monitoring [39,38,46]	No	BPEL-based orchestrations	Post-mortem	Functional Non-Functional	Instance
Lazovik [35]	No	Proprietary orchestration based deliver framework	Errors discovered as soon as they occur	Functional	Instance
Colombo [21]	No	Optimized middleware for SOA that supports BPEL	Before a message leaves the system, or before the incoming message is processed.	Non-Functional	Instance-Class
Cremona [37]	No	No specific paradigm, but any interaction between a caller and the provider	Re-active approach	Functional Non-Functional	Instance-Class
Glassfish [31]	No	Proprietary deployment Infrastructure	No automatic analysis. Timeliness does not depend on the system	Mainly non-functional	Instance
BP-Mon [12]	No	BPEL-based orchestrations	Signal information of interest as soon as they occur	Functional Non-Functional	Instance
Astro [7]	No	BPEL-based orchestrations	Signal information of interest as soon as they occur	Functional Non-Functional	Instance-Class
Extended Monitoring	No	BPEL-based orchestrations	Signal violations of interest as soon as they occur	Non-Functional	Class

Table 7.1: Comparison Table

monitoring properties the time they occur and not after the termination of the instance. Moreover approaches which perform monitoring by weaving code that implements the required checks inside the code of the system that is being monitored are concerned as intrusive approaches. Furthermore the kind and the scope of the monitored information is provided. The former one refer to the functional and non functional properties of a SBA while the latter one to instance or class application of the approach. Finally, the last property refer to the collaboration paradigm, it's monitoring approach use.

Chapter 8

Conclusion and Future Work

Our work focuses on monitoring the QoS compliance of Web services.

In this master thesis, we present a framework regarding to the problem of monitoring web services described as BPEL processes. The main component of our framework is Astro's WS-MON. Based on this component we implement a framework responsible for providing Astro with the necessary input files and also exploits the output results of this monitoring tool in order to provide the user with more monitoring properties. The input files needed for Astro are: the **abstract BPEL** processes, the corresponding **WSDL documents** and also a **choreography** (.chor) file, which contains the compositions's partners and also the definition of the monitoring properties. Furthermore, our framework checks at runtime the results of the monitors and reports the occurred violations. Finally, a specific case study is used to illustrate its functionality and to test the monitoring framework.

We believe that the main issue remaining for future work is to focus on the adaptation actions that must take place after a violation is detected. The dynamic and ever-changing nature of the business and physical environment requires Web services to be highly reactive and adaptive to the changes and variations they are subjected to. They should be equipped with mechanisms to ensure that they can adapt to meet changing requirements. A possible following future direction is the creation of a framework that is able to detect monitored events and derive suitable adaptation strategies.

Bibliography

[1] A. ALBRESHNE, P. FUHRER, and J. PASQUIER. Web services orchestration and composition. 2009.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, et al. Business process execution language for web services, 2003.

[3] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (ws-agreement). In *Global Grid Forum*, number GFD. 107, pages 1–47. The Global Grid Forum (GGF), 2004.

[4] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, et al. Web service choreography interface (wsci) 1.0. *Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems*, 2002.

[5] A. Arkin et al. Business process modeling language. *BPMI. org*, 2002.

[6] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Malhotra, et al. Web services policy framework (ws-policy). *Specification, IBM, BEA, Microsoft, SAP AG, Sonic Software, VeriSign*, 2004.

[7] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *Web Services, 2006. ICWS'06. International Conference on*, pages 63–71. IEEE, 2006.

[8] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. A timed extension of wscol. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 663–670. IEEE, 2007.

[9] L. Baresi, C. Ghezzi, and S. Guinea. Smart monitors for composed services. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 193–202. ACM, 2004.

[10] L. Baresi and S. Guinea. Towards dynamic monitoring of ws-bpel processes. *Service-Oriented Computing-ICSOC 2005*, pages 269–282, 2005.

[11] L. Baresi, S. Guinea, and P. Plebani. Ws-policy for service monitoring. *Technologies for E-Services*, pages 72–83, 2006.

[12] C. Beeri, A. Eyal, T. Milo, and A. Pilberg. Monitoring business processes with queries. In *Proceedings of the 33rd international conference on Very large data bases*, pages 603–614. VLDB Endowment, 2007.

[13] A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Siméon. Xml path language (xpath). *World Wide Web Consortium (W3C),*, 2003.

[14] M. Blow, Y. Goland, M. Kloppmann, F. Leymann, G. Pfau, D. Roller, and M. Rowley. Bpelj: Bpel for java technology (bpelj). *IBM developerWorks website, http://www.ibm. com/developerwork/library/specification/ws-bpelj*.

[15] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture, w3c working group note, 2004. *URL: http://www.w3. org/TR/2004/NOTE-ws-arch-20040211*.

[16] D. Box, F. Curbera, et al. Web services addressing (ws-addressing), 2004.

[17] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, and D. Winer. Simple object access protocol (soap) 1.1, 2000.

[18] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (xml). *World Wide Web Journal*, 2(4):27–66, 1997.

[19] M. Chen, A.N.K. Chen, and B. Shao. The implications and impacts of web services to electronic commerce research and practices. *Journal of Electronic Commerce Research*, 4(4):128–139, 2003.

[20] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, et al. Web services description language (wsdl) 1.1, 2001.

[21] F. Curbera, M.J. Duftler, R. Khalaf, WA Nagy, N. Mukhi, and S. Weerawarana. Colombo: Lightweight middleware for service-oriented computing. *IBM Systems Journal*, 44(4):799–820, 2005.

[22] F. Curbera, Y. Goland, J. Klein, F. Leymann, S. Weerawarana, et al. Business process execution language for web services, version 1.1. 2003.

[23] N. Delgado, A.Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *Software Engineering, IEEE Transactions on*, 30(12):859–872, 2004.

[24] V. Deora, J. Shao, W. Gray, and N. Fiddian. A quality of service management framework based on user expectations. *Service-Oriented Computing-ICSOC 2003*, pages 104–114, 2003.

[25] E. Di Nitto, V. Mazza, and A. Mocci. Collection of industrial best practices, scenarios and business cases. *S-Cube Consortium, Deliverable CD-IA-2.2*, 2:29–05, 2009.

[26] E.A. Emerson. Temporal and modal logic. *Handbook of theoretical computer science*, 2:995–1072, 1990.

[27] A. Endpoints. Activebpel open source engine, 2007.

[28] C.C.C. Federation, C.E. Law, and P. Probe. Advancing environmental health in child care settings. 2010.

[29] Y. Gao. Bpmn-bpel transformation and round trip engineering. *URL: http://www. eclarus. com/pdf/BPMN BPEL Mapping. pdf*, 2006.

[30] J. Hielscher, A. Metzger, and R. Kazhamiakin. Taxonomy of adaptation principles and mechanisms. *S-Cube project deliverable*, 2009.

[31] H. Hrasna. Glassfish community building an open source java ee 5 applica-tion server. 2006.

[32] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. *W3C Working Draft*, 17:10–20041217, 2004.

[33] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. Ws-bpel extension for people–bpel4people. *Joint white paper, IBM and SAP*, 2005.

[34] K. Kritikos and D. Plexousakis. Owl-q for semantic qos-based web service description and discovery. In *Proceedings of the SMR2 2007 Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web*, pages 123–137. Citeseer, 2007.

[35] A. Lazovik, M. Aiello, and M. Papazoglou. Associating assertions with business processes and monitoring their execution. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 94–104. ACM, 2004.

[36] G.T. Leavens, A.L. Baker, and C. Ruby. Preliminary design of jml: A behavioral interface specification language for java. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–38, 2006.

[37] H. Ludwig, A. Dan, and R. Kearney. Cremona: an architecture and library for creation and monitoring of ws-agreents. In *Proceedings of the 2nd International Conference on Service oriented computing*, pages 65–74. ACM, 2004.

[38] K. Mahbub and G. Spanoudakis. Run-time monitoring of requirements for systems composed of web-services: Initial implementation and evaluation experience. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pages 257–265. IEEE, 2005.

[39] K. Mahbub and G. Spanoudakis. Monitoring ws-agreements: An event calculus-based approach. *Test and Analysis of Web Services*, pages 265–306, 2007.

[40] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al. Owl-s: Semantic markup for web services. *W3C Member submission*, 22:2007–04, 2004.

[41] U. OASIS. Universal description, discovery, and integration (uddi) 2.0. *Organization for the Advancement of Structured Information Standards, Boston, MA (http://www.uddi.org)*.

[42] Tech. Rep. OMG. Business process modeling notation (bpmn) specification, final adopted specification. *Feb 2006, www.bpmn.org*.

[43] C. Ouyang, M. Dumas, W.M.P. Aalst, A.H.M.T. Hofstede, and J. Mendling. From business process models to process-oriented software systems. *ACM transactions on software engineering and methodology (TOSEM)*, 19(1):2, 2009.

[44] M. Papazoglou. *Web services: principles and technology*. Addison-Wesley, 2008.

[45] M. Shanahan. The event calculus explained. *Artificial intelligence today*, pages 409–430, 1999.

[46] G. Spanoudakis and K. Mahbub. Requirements monitoring for service-based systems: Towards a framework based on event calculus. In *Automated Software Engineering, 2004. Proceedings. 19th International Conference on*, pages 379–384. IEEE, 2004.

[47] S.A. White. Introduction to bpmn. *IBM Cooperation*, pages 2008–029, 2004.

[48] M. Wright and A. Reynolds. *Oracle SOA suite developer's guide*. Packt Publishing, 2009.