Customization of business process monitoring

Rafael Martinez, R.I.

Award date:
2013

Department of Mathematics and Computer Science
Department of Industrial Engineering & Innovation Sciences

Customization of Business Process Monitoring

Master Thesis

Ruben Ivan Rafael Martinez

Supervisors:
Dr. Marco Comuzzi and Dr. Giovanni Acampora
Dr. Ton Weijters
Dr. Christian Stahl

Final version

Eindhoven, July 2013

Abstract

One system that has demonstrated real advantages to the industry since the 1990s is the Workflow Managemen System (WFMS). A WFMS is basically a software that is in charge of managing a workflow execution by interpreting process definitions, identifying involved resources, interacting with workflow participants, and invoking external applications if required. It is important to state that workflow is a part of what is known as Business Process Management (BPM). BPM technology involves the creation, integration, automation and optimization of processes inside and outside an enterprise. One of the most important phases of Business Process Management is Business Process Monitoring because companies can use not only business process indicators, but also monitoring tools to have an efficient management of both business resources and business processes. Despite the importance of business process monitoring and the efforts that have already been implemented in the industry such as the reference model proposed by the WFM Coalition, it is true not only that the market is composed by different types of customers, but also that each customer has different necessities and expects different data from a business process execution environment. In order to address this situation we will use the term "Mass Customization", which under this context is defined as the ability of customers to customize the way in which providers execute a process [6]. It is necessary to clarify that customization involves a great diversity of aspects such as process control customization, where customers choose the activities that will be performed, or QoS customization where customer defines the level of security over resources, transactions or processes, etc. The management aspect is one of areas where it is possible to find a gap because BPM and customization have not been extensively investigated by the literature, and this is the main reason that the question "is it possible to find a generic customizable model for monitoring a business process?" becomes so important for this thesis. This project aims to find the answer for this question by defining a model, and taking this model from the conceptual design to the real implementation in order to prove if the model can be applicable in the real world.

Preface

This report is my master thesis for the conclusion of the Business Information Systems master program at Eindhoven University of Technology. Moreover, it is also the conclusion of my final project, which is related to the Business Process Management, one of the main concepts I learned in the master program. I really appreciated many people who helped me at the project.

I would like to extend my gratitude to Dr. Marco Comuzzi. I am happy that he supervised my thesis project because without him, I could not have dealt with this project. He gave me not only a lot of detailed instructions on my project but also many useful practical guidelines about how to implement a research project. All our meetings were really productive since he always had a clear picture of what he expected for this master thesis. In addition, Dr. Giovanni Acampora as my second supervisor of this project he gave me also a lot of constructive comments over data in the report. I really want to point out that even when both decided to change the course of their professional careers, they always demonstrated availability, patience and a great attitude towards my project. Besides, I would like to thank Dr. Ton Weijters and Dr. Christian Stahl, both members of my graduation committee, because they not only took time to evaluate my project, but will share their knowledge in order to make me aware of the advantages, and most importantly, of the possible opportunities of improvement in my approach.

Last but not least, I really appreciated my family, particularly my parents and my sister Claudia. Despite that she experienced a serious health problem in the period of my master program, she has showed great courage to face her life and to show me that we need to fight for the things that we want. This was one of the main reasons that helped me to complete this thesis project. Besides, I also want to thank to all the international people that gave me their friendship while I was living in the Netherlands.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter the problem area, the research question and the research approach used will be described.

1.1 Problem Area

Workflow Management Systems (WfMS) were created mainly to improve the automation and optimization of business operations by supporting the workflows through the use of state machines that helped to control the process flow from one activity to other [2]. In the 1990s when the WfMS systems became popular many vendors decided to provide several products. The great diversity of WfMS improved the industry, however, this also became a problem since each vendor created a product according to his idea of what should be the best for the customers, resulting in the creation of silos between and inside companies since the WfMS products were not compatible with each other. A group of companies joined together and created the WFM Coalition, this group addressed the problem by recognizing the common characteristics of the products and developing a reference model that gave a level of interoperability that was not possible before [16].

Even when workflow technology was already a great improvement for the industry, IT experts revealed that besides automation of processes, there were also other aspects that needed special attention from the industry. One of these aspects was the integration of systems and processes which later became one of the major challenges and goals for IT departments. This was the origin of what is called "Business Process Management" (BPM), a more generalized concept that considers the WfMs in its scope. It is necessary to state that even when BPM involves a wide range of aspects that are important for the industry, models, such as the WFMC reference model, continued as one of the key guidelines in the recent years.

So far it is clear to see that the use of new BPM technologies and reference models has improved the industry, however, we need to consider that this situation has also produced an increment in the number of technology customers, which is not a simple aspect since each customer has different necessities and expects different outcomes. In this particular field we find the concept of Mass customization, which according to [30] "relates to the ability to provide individually designed products and services to every customer through high process flexibility and integration". Mass customization has demonstrated several benefits such as the capacity not only to increase profit and decrease costs, but to develop a certain level of loyalty among customers because of its flexibility to provide what customers need.

On the basis of the importance of customization it is clear that even when the use of models in the BPM context has been a key step to evolve the situation of the market, the real question is how customizable are these models?. This question is difficult to answer but what is easy to know is that the degree of accuracy of the original WFMC reference model, one of the most representative models in the BPM context, was not enough, and that when we see the management aspect of BPM and customization we identify a gap because they have not been extensively investigated by the literature; furthermore, when we focus in the particular area of "Administration and Monitoring tools" we can see that each customer may have different requirements about monitoring, e.g. about what variables need to be monitored or about how the information is provided, the reality is not only that current technology provides limited options for business process monitoring customization, but also that process enactment technology provides standard consoles with limited access to the information, however, none of these options represents a generic model that can demonstrate easily which are the basic relevant variables and options that can be monitored in a BPMS. In the real world, a customizable model for business process monitoring will be useful since it will be a clear reference about what kind of monitoring information can be found in any BPMS, allowing customers to identify easily the type of information that can be used in order to define relevant KPIs and improve processes, situation which has a direct impact in the financial aspects of an organization.

1.2 Problem statement

The goal of this thesis is to identify both the basic functions that a monitoring tool should provide and the basic relevant data that should be collected. Once this information is gathered a new customizable monitoring model for business processes will be defined and finally it will be tested in a real environment with a real implementation.

The problem definition for this thesis is:
Is it possible to define a customizable monitoring model for business processes?

To answer this research question the following questions need to be addressed:

- why do customers need monitoring customization?
- what do customers expect from a customizable monitoring tool?
- In case there is a customizable monitoring model, is it possible to adapt it to a real environment?

1.3 Expected Outcomes

The expected outcomes of the project are:

- An analysis of the data that can be collected from a workflow engine.
- The proposed monitoring model for business process.
- An implemented tool where the model is proved in a real environment.

1.4 Scope

Since workflow administration and monitoring is a very wide topic we have to define a scope for this project.

The scope of workflow administration data will be limited to:

- The data that can be collected from a workflow engine directly.
- Process mining data will not be considered.

1.5 Criteria

We have the following initial criteria for the model and for the implemented tool:

- The model should demonstrate clearly the customization specifications that can be obtained.
- The model should consider the basic and relevant metrics that can be gathered in a BPMS.
- The tool should be able to demonstrate the flexibility of the monitoring model.
- The tool should work on a role based schema or at least provide a basic user control access.
- The tool should provide some standard reports to give this overview quickly.

1.6 Research Approach

The research plan aims at achieving the following objectives:

- **Understanding what kind of data shall be considered from a workflow engine** This will be done by a literature study.
- **Understanding what are the basic reporting aspects that a monitoring tool should support.** This will be done by a literature study.
- **Definition of the monitoring model** First, the results of the literature study will be used in order to define a set of monitoring variables and monitoring options. Once this set is defined, a new monitoring model will be built over it.
- **Implementation of the tool to prove the model** First, a conceptual architecture for the tool that proves the model will be defined, then a mapping between the conceptual architecture and a real technical architecture will be presented. Once this mapping is ready, the technical architecture will be used to implement a tangible tool that demonstrate the benefits that the model provides in the real world.
- **Evaluation of the tool** Software validation and verification aspects will be addressed.
- **Presentation of a complete conclusion about the results and the possible future work**

1.7 Outline

The rest of this thesis is organized as follows:

- **Chapter 2** Literature Study
- **Chapter 3** A Theoretical Model for Business Process Monitoring
- **Chapter 4** Implementation of the Model
- **Chapter 5** Evaluation of the Tool
- **Chapter 6** Conclusions
- **Appendix A** Information about BPMonitor

Chapter 2

Literature Study

On the basis of the scope and criteria mentioned in Chapter 1, we review literature that discusses the basics concepts related to Business Process Monitoring and Customization. Besides we also review literature that discusses Reporting Techniques.

2.1 Business Process Monitoring

One of the key concepts that is addressed in this thesis report is "Business Process". According to [31] a business process is defined as "coordinated chain of activity intended to produce a business result.". With this notion, it is possible to see that simple processes can be inside the scope of an organizational unit, however, end-to-end business processes can go not only from one department to another, but also from one business partner to another. Basically, it is possible to find several business processes in a company and their performance usually determines the grade of success of the company.

In the 1990s Hammer and Champy came up with the idea that redesign of business process was the key factor for companies to lower costs and become competitive. This idea was the basis of Business Process Reenginering (BPR) [14]. BPR was implemented under the simple concept that IT could help organizations to achieve dramatic changes that will let them to be better and more successful. Even when it is true that IT can help organizations to make a critical and aggresive change, there were some problems related to BPR. The reason of these problems was that rather than making incremental improvements the focus was to make large and difficult changes which led to complex projects that in the end were difficult to manage and implement.

Some years later in the 2000s the concept of Business Process Management (BPM) came into scene and became popular within the next years. The reality is that there are many definitions of BPM in books, articles, conferences and web sites around the world. So while BPM can have several meanings I decided to provide some definitions that are relevant for this thesis report. The first definition is from the IBM website where BPM is defined as follows: "BPM is a discipline that leverages software and services to provide total visibility into your organization. Discover, document, automate, and continuously improve business processes to increase efficiency and reduce costs" [19]. Checking other definitions it is clear to see that there is still some uncertainty about recognizing BPM as a technology or as a discipline and to clarify the concept I decided to include as a second definition the one of the book written by Jeston and Nelis where it is stated that "BPM does not equate to a technology tool or initiative for business processes. In our experience, there is significant business process improvement that can be achieved without technology." [21]. Finally, the last definition is taken from the Association of Business Process Management Professionals (ABPMP). ABPMP is a non-profit organization that is vendor independent and that is dedicated to the advancement of business process management concepts and its practices. For this organization, Business Process Management (BPM) is defined as follows: ”a disciplined approach to identify, design, execute, document, measure, monitor, and control both automated and non-automated business processes to achieve consistent, targeted results aligned with an organizations strategic goals. BPM involves the deliberate, collaborative and increasingly technology-aided definition, improvement, innovation, and management of end-to-end business processes that drive business results, create value, and enable an organization to meet its business objectives with more agility. BPM enables an enterprise to align its business processes to its business strategy, leading to effective overall company performance through improvements of specific work activities either within a specific department, across the enterprise, or between organizations” [3]. The last definition is the most complete since it gives a clear idea of BPM and its importance within an enterprise. Recent studies show that most information technology executives consider BPM as the most important technology that help them achieve their business goals [32].

As it was detailed in the previous definitions, BPM involves a series of phases which can be depicted in a life-cycle. According to [2] the BPM life-cycle can be depicted as in figure 2.1. Now a brief description of the life-cycle is provided [25]:

Design phase
: This phase is to create a process or an alternative one when there is already an existing process.

Configuration phase
: This phase specifies all elements needed to implement the process definition created in the design phase.

Execution phase
: In this phase the configured workflow becomes operational by transferring the process definition to a workflow engine.

Control phase
: This phase is to monitor the execution of the operational business process.

Diagnosis phase
: This phase is to discover any weakness in the process on the basis of the information gathered in the control phase.

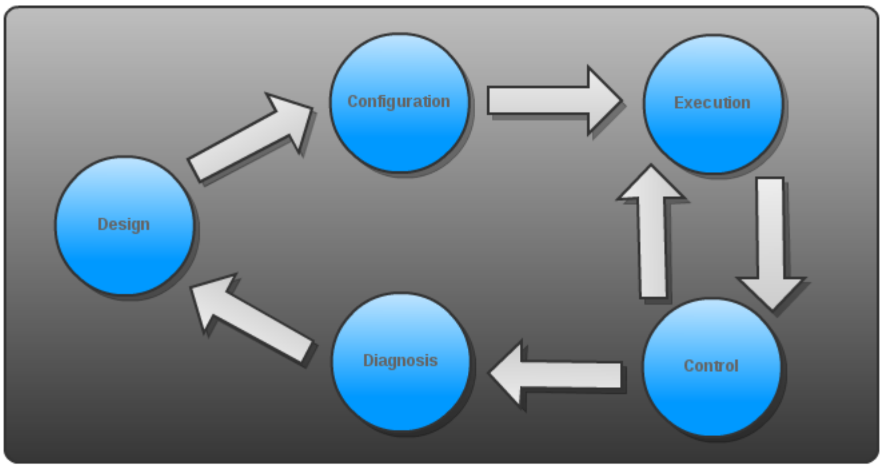


Figure 2.1: BPM life-cycle

This life-cycle clearly reflects the usual flow followed in many organizations where the business analysts design a business process model, which is then refined by IT engineers to a model that can be executed. The executable process model is deployed to a process engine, which executes the process by delegating tasks to humans and services. Once the process is executed we find aspects, such as continuous supervision of both business goals and business process performance, which are related to what is known as business process monitoring.

When we are talking about monitoring it is necessary to clarify some important concepts such as: ”Process Measurement”, ”Process Monitoring” and ”Business Activity Monitoring”. According to [22] these terms are defined as follows:

- **Process Measurement:** It is defined as the action to determine the performance of business process using a provided set of performance indicators.
- **Process Monitoring:** It is defined as the action to take a careful look over a specific situation to reveal something about it.
- **Business Activity Monitoring:** According to the Gartner Group, Business Activity Monitoring is defined as follows [10]: ”BAM is the real-time reporting, analysis and alerting of significant business events, accomplished by gathering data, key performance indicators and business events from multiple applications.”

This thesis will use the concept of ”Process Monitoring” as its basic guideline.

2.1.1 Customization

In the recent years several industries are facing the issue that customers demand both that companies fulfill their orders as fast as possible and that companies can offer highly customized products and services. The reality has demonstrated that companies have already found difficulties to fulfill orders or services in a considerable time and at an acceptable cost without considering the customizable aspect. This kind of challenges made that concepts such as ”Mass Customization” came into scene. This concept is simple, it basically means to provide personalized products at reasonable prices.

In the literature it is possible to find several references to mass customization, but we decided to use the reference of [26], which is one of the first resources which popularized the concept. In this paper we find the five elemental methods of achieving mass customization for a standard product producer:

- customize services around standard products
- create customizable products
- provide point of delivery customization
- provide quick response
- modularize components

According to Pine, the author of the paper, any company that wants to change to a mass customization approach has to follow these same steps or at least a similar flow in this order. So far we have analyzed what is the main idea of mass customization and the basic steps to introduce the approach into a real case, however, we need to consider if customization is really important for the customer?. When we check other sources of information we find that mass customization may create a value for customers by closely matching a product to his or her necessities. In this paper [33] it is stated that ”Because value originates in a person’s needs system (Connor and Davidson 1985), a product closely matching those needs may be perceived to be of higher value. Firms have known this for centuries, offering bespoke solutions to consumer needs”.

Now that we have a clear definition of both Business Process Monitoring and the importance of customization we need to define the link between these two concepts.

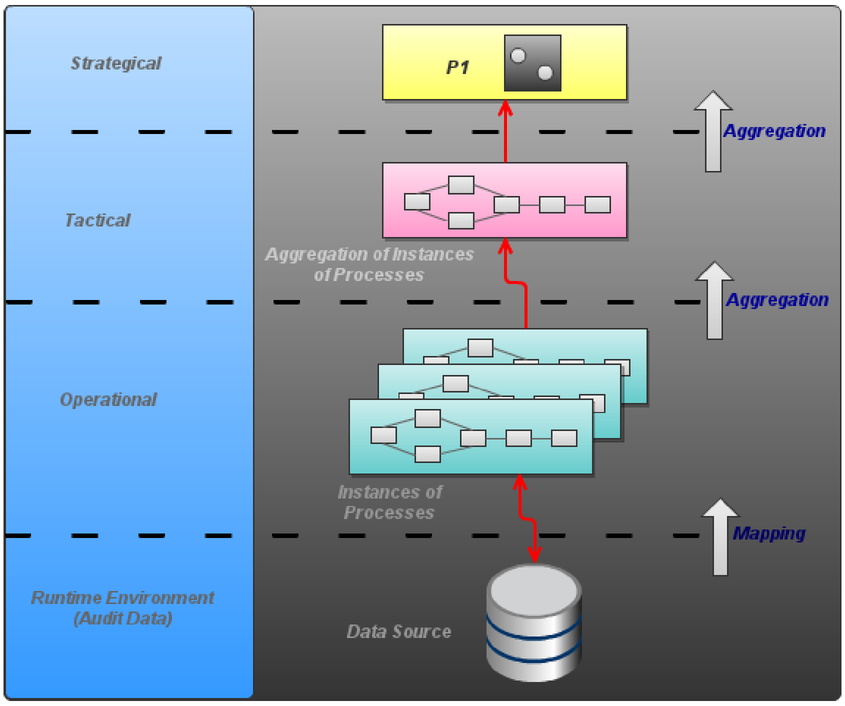


Figure 2.2: Management levels and the Business Process Execution Environment.

When we analyze the situation in the real world we can find different levels of management in any organization. According to the literature these levels can be classified as strategic, tactical and operational. The paper [29] states that "The strategic level designs the logistics network, including prescribing facility locations, production technologies and plant capacities. The tactical level prescribes material flow management policies, including production levels at all plants, assembly policy, inventory levels, and lot sizes. The operational level schedules operations to assure in-time delivery of final products to customers". This definition is useful because it indicates the main differences between the levels.

In the paper [24] we find that when we focus on process-based, and particularly on workflow-based monitoring, all of the management levels use information related to the business process execution environment, situation depicted in figure 2.2. This is a clear indication that inside a company we find the customers of the process monitoring systems. Moreover, these customers may use monitoring information to perform evaluations and improvements over processes, resources, suppliers, etc., and they are likely to have specific monitoring requirements about both what information should be available and how the information is accessed and presented. This kind of arguments are supported by papers such as [6] where it is stated that "Customization of monitoring requires focus on both control flow aspects, specifying the way to capture monitoring information and make it available, and resources, specifying what has to be monitored and at which stage of the process". This statement reflects the basic idea of the possible features that need to be considered by our monitoring model.

It is necessary to clarify that in order to offer a set of customization specifications the first step is to define what are the possible values, features, concepts that can compose these specifications. In the specific case of process monitoring we first need to define what are the business process metrics can be gathered from a BPMS. In the next section we will review literature about these metrics.

2.2 Business Process Metrics

So far the concepts of Business Process Monitoring and Customization have been addressed, however, we still need to define the data that should be gathered from the workflow engine. The term of ”metric” will be used to make a reference to the data that will be gathered.

We start by taking into account the paper of [12] because it has been the main reference for several papers that were written in the recent years. This paper is related to Business Process Intelligence (BPI), which is defined as ”a set of integrated tools that supports business and IT users in managing process execution quality”. Even when BPI is out of the scope of this thesis, the reality is that this paper is one of the main references where we find monitoring in a business process context. Indeed, according to this paper monitoring is defined as follows: ”BPI can monitor and analyze running process instances, and inform the user of unusual or undesired situations. Users can view the health status of the system, processes, services, and resources. In addition they can define critical situations (alerts), so that BPI can notify them on the medium of their choice in the event such a critical situation occurs”. It is clear that the main idea of the monitoring definition is closely related to the one we defined in the previous section. Once this relationship is stated, we can read in a more detailed way through this paper and we will find that the main idea is based on the construction of a BPI tool over HP Process Manager (HPPM), which is a BPMS. Basically the BPI tool will perform a series of tasks over the HPPM and these tasks will provide important information to the final users of the system. The main point for us is to understand how this tool is connected to the BPMS and what kind of data can be gathered from the BPMS.

The paper states that the BPMSs logs information on process, node, and service executions into an audit log database, which is typically stored in a relational DBMS. This database is a crucial element for the connection between the BPMS and the BPI tool, so it is the main point that we will analyze. In the figure 2.3 is depicted the database schema of the HPPM audit log [12].

It is clear that we do not need to understand the complete architecture of HPPM in order to recognize the kind of data that is stored by the BPMS. In this case we can see two important aspects. The former is that according to the database schema the information is distributed in different tables to store information about several levels of detail such as node or service. The latter is that the basic elements that we see are time related data, state related data and resources related data. According to this notion we can have the first version of the metrics that have to be considered in our model and this is presented in table 2.1.

When we take into consideration other sources of information such as the paper [9] we can see that even when the paper is focused in the use of Business Process Modeling Notation (BPMN) for Business Activity Monitoring, and not really for Business Process Monitoring as was defined for this thesis, in a general point of view the paper is also addressing the concept of monitoring and presenting a concrete syntax to model monitoring activities with the use of a powerful modeling

BPMS	Time Data	State Data	Resources Data	Levels of information
HPPM [12]	Activation Time, Completion Time	State	Initiatior, ExecutedBy	Node, Service, Process

Table 2.1: Metrics Gathered from BPMS Engine First Version

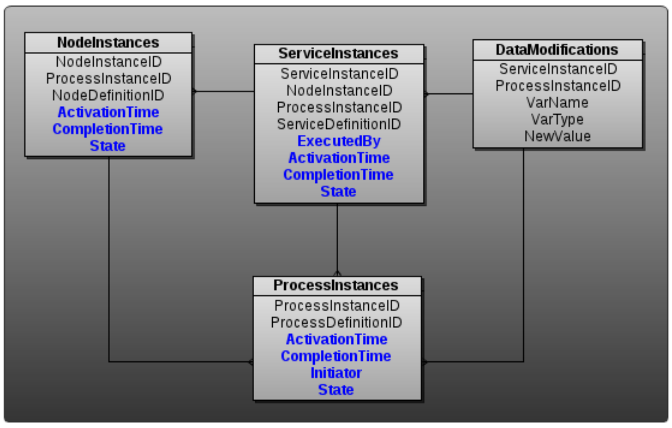


Figure 2.3: Database schema of the HPPM audit log

BPMS	**Time Data**	**State Data**	**Resources Data**	**Levels of information**
HPPM [12]	Activation Time, Completion Time	State	Initiatior, ExecutedBy	Node, Service, Process
Generic BPMS [9]	Duration	State Ocurrence		Basic Instance Measures, Process Measures, Basic Measure, Composed Basic Measure, Aggregated Measure

Table 2.2: Metrics Gathered from BPMS Engine Second Version

notation such as BPMN. A careful review of the proposed BPMN syntax was performed and the main points for us are presented in the figure 2.4. This figure shows again a separation of the information in different levels of detail. This paper is really interesting since it is not only separating information in ”Basic Instance Measures” and ”Process Measures”, but making a distinction between a basic metric such as ”Duration”, a ”Composed Basic Measure” and an ”Aggregated Measure”. Besides the different levels of detail that are presented, this paper also mention ”Duration”, which is a time data, and ”State Ocurrence”, which is a state data. After the analysis of the second paper [9], it is possible to update the information about the metrics that have to be considered in our model and this is presented in table 2.2.

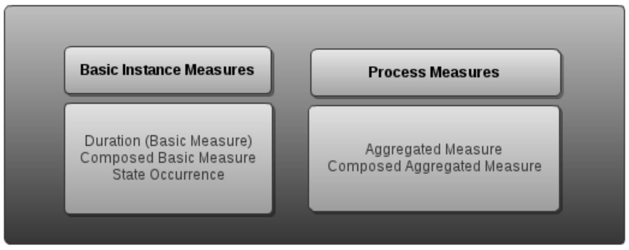


Figure 2.4: BPMN syntax for monitoring

Finally, we will take into account the information of one last source of information. This last source is the paper [34] where it is possible to find a section focused on the construction of elementary process metrics. In this section we can find the following statement: "The most elementary process metrics are obtained by analyzing the time-stamps of several process-related events that belong to the same process or activity instance. The difference between these time-stamps can provide an analyst with some basic insights into the behavior of a process instance. At the most basic level a process management system delivers frequency and temporal information to decision makers". It is clear again that temporal information is not only one of the basic metrics in a BPMS, but also one of the most important when we are talking about monitoring.

This paper uses as a reference the model defined by the WfMC in the specification Business Process Analytics Format - Draft Specification. 1.0, which is depicted in the figure 2.5. This model is really important since it shows the great variety of states that will be traversed by a business process or a business process activity over the lifetime of the respective process or activity instance.

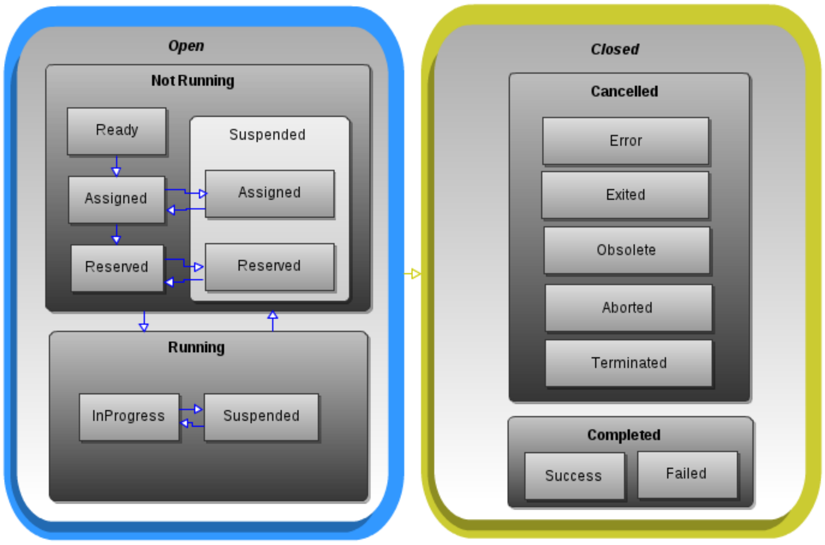


Figure 2.5: Process Execution State Model defined by the WfMC

The model has two main superstates *Open* and *Close*. According to the model, an activity or process instance in the state *open* can change its state, while and activity or process instance in the state *close* has arrived to its terminal state. Besides, these two superstates are divided into a number of sub-states and some of their definitions are presented below:

- **Open.NotRunning** This state means that no work is performed, but there is a possibility that the instance is assigned or reserved by some resource.

- **Open.Running** This state means that a process or an activity instance is currently in process.

- **Close.Cancelled** This state means that a process or an activity instance was force to terminate. There are several reasons for this state such as an unexpected system error, a manual cancelation or just because an activity simply became obsolete. It is important to clarify that this state means that the process or activity instance did not terminated as it was supposed to and that it is also true that it did not complete its goal.

- **Close.Complete** This state means that a process or an activity instance was fully processed. Despite the fact that the activity or process instance terminated correctly, it is not guaranteed that it has achieved its goal. In order to know if the goal was achieved or not we find the states "Close.Complete.Success" and "Close.Complete.Failure".

- **Open.Assigned** This state describes the situation when the system places an activity instance in the work list of one or several system roles.

- **Open.Reserved** This state describes the situation when one of the users with the role to whom the activity was offered selects the work item.

- **Open.Running.InProgress** This state describes the situation when the user that selected the work item starts working on it.

- **Open.Running.Suspended** This state describes the situation when the user that is working on a work item suddenly decides to suspends and continue working on it later.

- **Closed.Completed.Success** This state describes the situation when the user that is working on a work item completes the activity instance.

It is important to state that even when the model shows a great diversity of states, not all of them can be found in every BPMS simply because not all the BPMS can support all the described operations.

It has been clear that in all the sources of information previously reviewed the temporal metrics are important and this model and their states and sub-states are useful to find more specific metrics related to time. These metrics are defined below:

- **Turnaround Time** Time between the moment an activity instance becomes ready for execution and the moment the user completes the activity instance.

- **Wait Time** Time between the moment an activity instance becomes ready for execution and the moment the user selects a work item.

- **Change-over Time** Time between the moment the user selects a work item and the moment the user starts working on it.

- **Processing Time** Time between the moment the user starts working on a work item and the moment the user completes the activity instance.

- **Suspend Time** Time between the moment the user suspends the activity instance and the moment the user resumes the activity instance and continues working on it.

After the analysis of the last paper [34], it is possible to update the information about the metrics that have to be considered in our model and this is presented in table 2.3.

BPMS	Time Data	State Data	Resources Data	Levels of information
HPPM [12]	Activation Time, Completion Time	State	Initiatior, ExecutedBy	Node, Service, Process
Generic BPMS [9]	Duration	State Ocurrence		Basic Instance Measures, Process Measures, Basic Measure, Composed Basic Measure, Aggregated Measure
Generic BPMS [34]	Turnaround Time, Wait Time, Changeover Time, Processing Time, Suspend Time	Open.NotRunning, Open.Running, Close.Cancelled, Close.Complete, Open.Assigned, Open.Reserved, Open.Running.InProgress, Open.Running.Suspended, Closed.Completed.Success		Activity instance, Process Instance

Table 2.3: Metrics Gathered from BPMS Engine Final Version

So far we have analyzed the literature which indicates what kind of metrics can be found in a BPMS and they are a well supported starting point about the type of information that can be considered by the model. However, as it was discussed previously a customizable model involves also some technical options such as the way the information is accessed. In the next section literature over this type of aspects will be reviewed.

2.3 Reporting Techniques

Since the origin of the Internet, users around the world have been using a very basic way to get the information and this way is well known as "Information Pull". This term means that the user decides when to get the information because he or she takes the role of the "initiator" and they make a request to the system. Over the years this way of information retrieval has been really popular and useful for the internet industry, however, this also brought some issues when both the internet user population and the diversity of services over internet increased because in some cases some aspects such as the response time of a service or even the availability of the service were affected [15]. Besides these issues, another aspect that has become important in the recent years is the necessity that systems respond not only to the user's input but to the user's environment as well [7]. All of these aspects gave origin to a different way to get the information. This way of information retrieval is known as "Information Push" and it means that the role of the "initiator" can be taken by more elements and not only by the user. In the specific case of information systems, the system is the one that can actually take this role and retrieve information to the user in a an automatic way without any request from the user.

Another important aspect that needs to be considered in a reporting system is the security when we retrieve the data. One of the most popular techniques to implement security in a system is to use Role-Based Access Controls. According to the paper [8] the use of roles in a system has several advantages because it is basically a way of restricting access to data based on the identity of subjects or groups to which they belong. Under this schema organizations have not only a good separation of activities between roles, but the certainty that specific tasks are performed by specific roles. If we analyze this idea it is possible to see that a role based system is basically affecting the content of what a user can or cannot see. With this kind of control it is possible for the system to offer options such as define if the required data will be available in the future or not. For the purpose of this thesis we will called this option as "consume" meaning that the data will not be available in the future while with the option "read" the data will be available in the future.

When we are talking about reporting techniques, we need to focus our attention not only to the way information is retrieved or the security aspects, but to the content itself that is retrieved and how this is managed by both the service that handles the request and by the client interface that displays the results to the final users. If we take a look of this article [20] we can see that it points out the necessity of considering historical data. According to this article historical data is important because it allows the user to reveal hidden information that cannot be found when users limited their vision only to the screenshot of the current situation. It states literally that "historical information is crucial to understanding the seasonality of business and the larger cycles of business to which every corporation is subject". Under this notion, a reporting system should at least two possible options. The former is to present the information of the current situation and the latter is to consider the option of managing historical information related to the requests of the user.

Chapter 3

A Theoretical Model for Business Process Monitoring

In this chapter we first discuss a generic conceptual model of the business process monitoring life-cycle. Then, we analyze this life-cycle model and the information of the literature study to define a set of monitoring variables and monitoring options. Finally we built the model for cross-instance business process monitoring over the defined monitoring set.

3.1 Business Process Monitoring Life-cycle

In the previous chapter a clear definition of Business Process Monitoring was provided, however, there is still missing a reference to the basic monitoring architecture that commonly supports business process monitoring. According to the paper [6], the general monitoring architecture is composed by the elements in figure 2.3. The architecture is composed by two main domains. The former is the "Customer Domain" and contains the module of "Monitoring Client" which is basically the interface with the users. The latter is called "Business Process Provider Domain" and contains two main modules "Monitoring Infrastructure" and "Business Process Engine". Monitoring Infrastructure module is usually called observer since its main function is to capture relevant information over latter module, which in general terms can be defined as a component which produces business process information such as process or activity instance states, times, data values, etc. It is necessary to clarify that even when the module of "Monitoring Infrastructure" can be built either directly over, or in some cases inside the "Business Process Engine", it is and should be taken as a different entity. With this notion, it is possible to have a better understanding of both the source of information and the monitoring component that will be in charge of providing relevant information to the user.

Once that we have a clear idea of the basic monitoring architecture it is possible to see the basic business process monitoring life-cycle which is described below:

1. The Monitoring Client triggers the monitoring.
2. The Monitoring Infrastructure receives and identifies the kind of request from the client.
3. The Monitoring Infrastructure obtains the required data from the Business Process Engine.
4. The Monitoring Infrastructure manages the data.
5. The Monitoring Infrastructure transfers the results of the request to the Monitoring Client.

In order to have a good understanding of the monitoring life-cycle it is important to clarify the following concept:

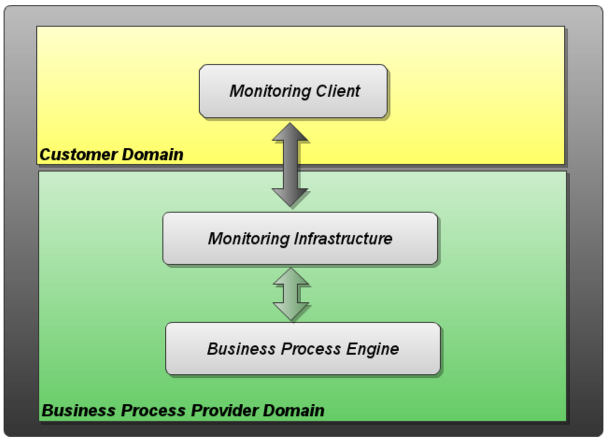


Figure 3.1: General Business Process Monitoring Architecture

Monitoring Variable (MV) We will call Monitoring variable to what we want to monitor. When we are talking about business process monitoring terms a MV can be used to reference several types of data, e.g., infrastructure-level data such as respond time, application-level data such as the state or processing time of an activity in a specific process instance [6]. The relationship between the term MV and the monitoring life-cycle is basically that the MV specifies the process information that the Monitoring Infrastructure obtains from the Business Process Engine and supplies to the Monitoring Client.

Even when the life-cycle that was presented can be understood in a straightforward way there are some details that have to be pointed out. In the case of the phases regarding the managing of the data and the transference of results to the client we will use some concepts that were also described in the section 2.3 of the previous chapter:

- Managing the data obtained from the Business Process Engine should consider to provide the options of either rewriting the obtained data for a specific monitoring variable, or to store the data in order to display not only the screenshot of the current situation, but also a historical view which according to our literature review could help the users to reveal relevant hidden information. Besides when we are talking about content it is necessary to consider other options such as the availability time of the data. This means that the user should be able to decide also if the data is "consumed", meaning that it will not be available in the future, or "read" meaning that it will be available the next time when the user requests the information about a specific monitoring variable.

- Transferring the results of the request to the Monitoring Client is a phase that should consider the options of how the requested data will flow from the service that process the request to the client, which is the interface with the final user. As it was described in the previous chapter, the two main approaches that are considered in this terms are "Information push" and "Information pull". The monitoring system should allow the user to choose what kind of option would be better for the situation.

In this section we explained the elemental architecture of Business Process Monitoring life-cycle, identifying some of the options that should be considered by the model. In the next section we will analyze the relationship between the monitoring life-cycle and the information obtained in the previous chapter to finally obtain the monitoring set of monitoring variables and options that will be used by the model.

Time Data	State Data	Levels of information
Activation Time, Completion Time	State	Node, Service, Process
Duration	State Ocurrence	Basic Instance Measures, Process Measures, Basic Measure, Composed Basic Measure, Aggregated Measure
Turnaround Time, Wait Time, Change-over Time, Processing Time, Suspend Time	Open.NotRunning, Open.Running, Close.Cancelled, Close.Complete, Open.Assigned, Open.Reserved, Open.Running.InProgress, Open.Running.Suspended, Closed.Completed.Success	Activity instance, Process Instance

Table 3.1: Considered Metrics from a BPMS Engine

3.2 Business Process Monitoring Variables

In the previous chapter we gathered information about both the basis of business process monitoring and the basic relevant metrics that can be obtained from a BPMS. Indeed, in the final version of the table with the basic relevant metrics 2.3 is possible to see four main elements: *Time data*, *State data*, *Resources data* and *Levels of information*. This table also shows that contrary to *Time data*, *State data* and *Levels of information*, *Resources data* is only found in the reference related to a specific BPMS (HPPM). This is really important for us since this means that there was no clear reference to resources information in the sources related to generic BPMS systems.

Since the purpose of this thesis is to develop a generic customizable monitoring model which can be applicable to any BPMS, it is clear that we need to focus on those metrics that are most likely to be present in any BPMS in order to ensure the success of our model. Taking this argument into account, we see that *Time data*, *State data* and *Levels of information*, depicted in table 3.1, are the most relevant metrics for us since they can ensure the applicability of our model at least from the theoretical point of view.

So far we have both a clear vision of what metrics should be considered from a BPMS and a solid definition of the business process monitoring life-cycle. However, we have not explained how these two aspects are linked. In the previous section we described the concept of "Monitoring Variable" and this is the key to find the link we are looking for between the metrics and the life-cycle model. It is necessary to point out that a MV is what we want to monitor and is important to know that a monitor variable can be monitored depending on a set of parameters that we will call "options" and the results will vary depending on the options selected. This means that a monitoring variable does not have a one to one relationship with the set of results it may have.

When we analyze the table 3.1 we can see that in a general point of view the metrics of "Time Data" and "State Data" can be two monitoring variables because they are indeed a valid target that a user may want to monitor. In the case of "Levels of information" it is clear that in a business process environment the detail of information is not a variable, but an option that can be chosen by the user to receive a different result for a monitoring variable.

If we analyze the metrics of "Time Data" we can see that the last row has the most representative types of time since "Activation Time", "Completion Time" and "Duration" are implicit in the set composed by "Turnaround Time", "Waiting Time", "Change-over Time", "Processing Time" and "Suspended Time". Each of these types of time are a different monitoring variable since each of them can have a value for which a user may be interested.

In the case of the metrics of "State Data" we can also see a great diversity of values in the three rows, but it is again in the last row where we find the most representative metrics related to state. With this notion, we need to point out that this case is different from the time related monitoring variables that we revealed previously because the state related metrics are not something that a user may be interested to know. On the other side, in a business process environment, users are usually interested to know the state of a process or activity instances, so the state is the monitoring variable and the metrics that we found in the table compose the state space for the possible values that can be taken by the monitoring variable of state.

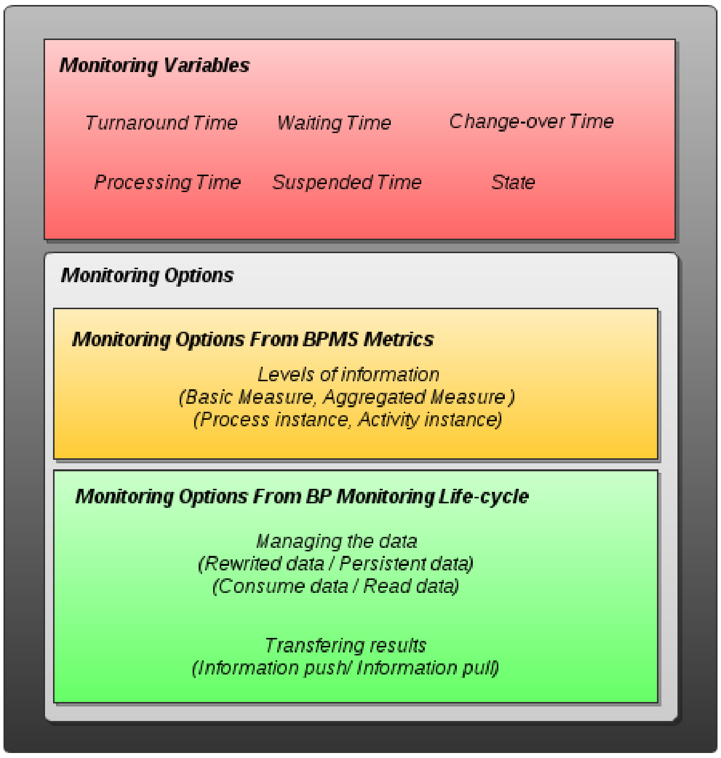

Monitoring Variables
Turnaround Time
Waiting Time
Change-over Time
Processing Time
Suspended Time
State
Monitoring Options
Monitoring Options From BPMS Metrics
Levels of information
(Basic Measure, Aggregated Measure)
(Process instance, Activity instance)
Monitoring Options From BP Monitoring Life-cycle
Managing the data
(Rewrited data / Persistent data)
(Consume data / Read data)
Transfering results
(Information push/ Information pull)

Figure 3.2: Monitoring Variables

After the analysis of both the metrics gathered from a BPMS and the business process monitoring life-cycle we have the set of the monitoring variables and monitoring options that will be the basis of our monitoring model. These values are depicted in figure 3.2. In the next section we will use these values in order to define our model.

3.3 Business Process Monitoring Model

Customization of business process monitoring is made by users specifying their preferences, or options, over different dimensions characterizing process monitoring. This is why the proposed model defines dimensions and options on the basis of the monitoring variables and monitoring options that were revealed in the previous section. The multi-dimensional space of monitoring dimensions and the related options of the proposed monitoring model are shown in the figure 3.3. Moreover, it has been stated that Customization of Monitoring is made by users making a choice over diverse monitoring dimensions, however, we still need to define how this choice is considered by our model. The concept that we are looking for is defined as "customizable configuration". A customizable configuration will specify the user choices about monitoring at a specific point in time. With this notion, it is clear that the multi-dimensional space of the model can be used to create a rich diversity of configurations.

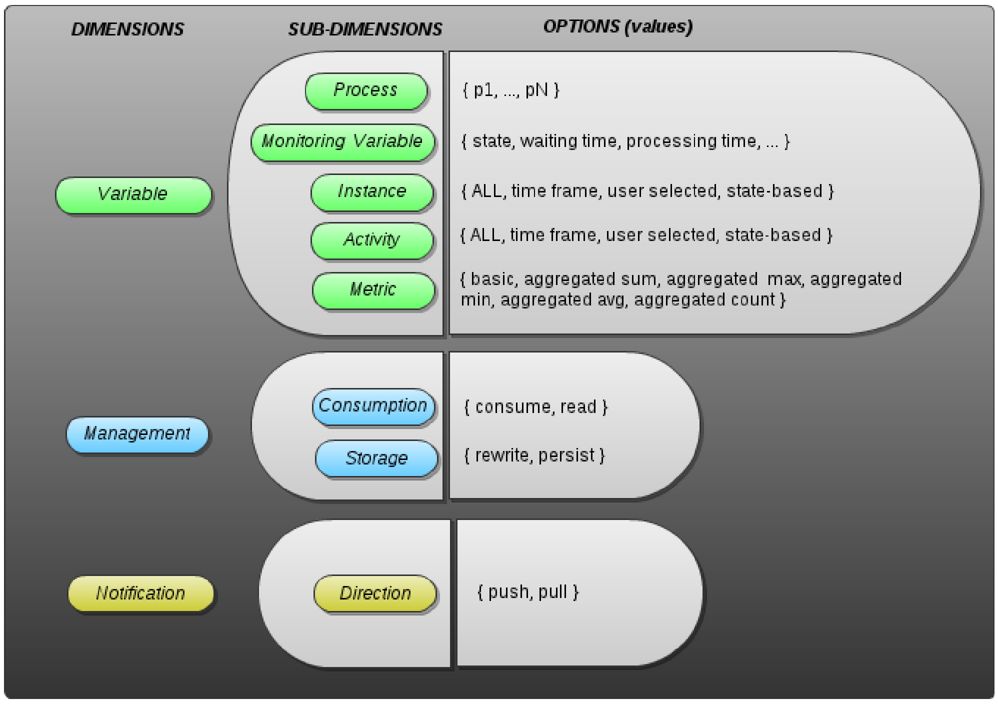

DIMENSIONS
SUB-DIMENSIONS
OPTIONS (values)
Variable
Process
{ p1, ..., pN }
Monitoring Variable
{ state, waiting time, processing time, ... }
Instance
{ ALL, time frame, user selected, state-based }
Activity
{ ALL, time frame, user selected, state-based }
Metric
{ basic, aggregated sum, aggregated max, aggregated min, aggregated avg, aggregated count }
Management
Consumption
{ consume, read }
Storage
{ rewrite, persist }
Notification
Direction
{ push, pull }

Figure 3.3: Multi-dimensional Space of the Monitoring Model.

The problem can be formally characterized as follows. We consider a set of I users U, with $U = \{u_1, ..., u_I\}$. A user u_i may have specified one or more customizable configurations $\{c_{i,j}\}$ (a customizable configuration captures the choice of the user). That is, the set of customizations currently specified in the system is:

$$C = \{c_{i,j}\}_{i=1...I, j=1...J}$$

A customization $c_{i,j}$ is a tuple comprising the element V (*Variable*), M (*Management*), and N (*Notification*):

$$c_{i,j} = \langle V, M, N \rangle$$

The V element in a customization specifies the options chosen in respect of the Variable dimension in the monitoring model (see the figure 3.3):

- *pro*: the process to be monitored.
- *var*: the monitoring variable to be monitored.
- *inst*: the instance(s) to be monitored.
- *act*: the activity(ies) to be monitored.

- *met*: the indication of either basic result or aggregation function over the retrieved monitoring values.

That is, V is a tuple $V = \langle pro, var, inst, act, met \rangle$, where the elements of the tuple can assume the values shown in figure 3.3. It is necessary to give a brief clarification of these values:

The process p is the identifier of the K processes currently deployed in the BPE, that is, $p \in \{pk\}$, with $k = 1, ..., K$. According to the analysis performed in the previous section, the monitoring variable is limited to time and state data. In the case of instance and activity we consider options that allow users to use time, state or a specific selection criteria of instances and activities respectively. About metric, we consider either the option to see the basic result of the selected monitoring variable, or an aggregated result, which is based under the specification of the SQL language aggregation operators definition, i.e. $met \in \{basic, \{sum, avg, max, min, count\}\}$. Note that not all combinations of options are feasible. For instance, it is not possible to apply aggregation operators such as *sum* or *avg* to non-numeric values, e.g. activity or instance state. Similarly, operators *max* and *min* can be applied only on monitoring variables for which a total order relationship is defined.

The M element specifies the options chosen in respect of the Management dimension, whereas the N element captures options in respect of the Notification dimension. There are two sub-dimensions for management, namely consumption C and storage S, that is, $M = \langle C, S \rangle$, with $C \in \{consume, read\}$ and $S \in \{rewrite, persist\}$. There is only sub-dimension direction D for notification, with $D \in \{push, pull\}$.

Use of the Model

In the recent years things have been changing rapidly because technology has connected the world, increasing the number of communication channels, offered services and types of customers. One of the areas that has faced great changes is the the banking industry simply because day after day more users decide to change their usual transactional operating mode from the physical world to the digital one, or simply because they decide to start handling their money in an online way.

Because of this situation, banks are aware of the need to ensure that their offerings not only enter in the market quickly with few errors, but also that they need to be competitive on price. In order to achieve this goal and respond quickly to the customer needs, banks know that their processes must be standardized and well orchestrated. This is where BPM systems came into scene for them.

A specific case is Wal-Mart Bank, a specialized retail bank that started introducing business process management (BPM) and service-oriented architecture (SOA) in 2011. The BPM system effectively connected the front-office to the back-end system of the bank, reducing the number of manual errors in the process, increasing the system transparency and optimizing the majority of the processes.

Being a retail bank means that order management process is one of the basic business processes for the Bank. A high level flow of the process is described as follows:

1. Front-office system captures basic customer information.

2. A contract is printed with an id that matches the order data and the archive document. The physical copy is signed and sent to headquarters.

3. Back office, employees follow a pre-determined set of processing guidelines to fulfill the request.

4. The process is complete within 40 hours of initiation.

This is a great scenario for us since we can find a real situation where our monitoring model can become a critical aspect for the business. In particular when we are talking about a financial institution, and in this case of a retail bank, which main characteristic is that it performs transactions directly with customers, and not with other businesses just as commercial banks do. With this notion, it is easy to see that the end of the year is one of the most important seasons for a retail bank since most of the people (possible customers) probably will need an account to make transactions for the season. Specially during this season, one of the main goals of the bank is to complete their order management process as fast as possible and this is where our model makes a perfect match. Let us consider that we are in the middle of December and a manager of operations department u_1 is interested to know what has been the performance of the employees that are involved in the order management process during the first two weeks of the season. One way to reveal data about his concern is to check how much time on average an employee takes to select a work item, or how much time takes him to start working on it. We can translate easily these requirements in the terms of our model. If we focus on the average time an employee takes to select a work item then we are talking about the average waiting time, and we can consider all activities for instances started after December 1st, 2013 of processes p1 where p1 is order management process. In this case we assume that the manager is interested to see the data when he requests it, but he also considers that he needs to see all the results that he requests in order to have clear evidence of how the changes he will be making in the process are going to impact the waiting time. These two arguments are also easily translated in our model since we know that he wants to pull the monitoring information and that he wants the monitoring values to be persisted and not destroyed after being requested. As it was previously described we will capture the choice of the user into a "customizable configuration". This configuration is $c_{1,1} = \langle V, M, N \rangle$ where:

$$V = \langle p1, WaitingTime, \geq \text{2013/12/01}, ALL, avg \rangle$$

$$M = \langle persist, read \rangle$$

$$N = \langle pull \rangle$$

In this chapter we defined the multi-dimensional space of both monitoring dimensions and related options of the proposed model. Besides a theoretical definition and an example with the application of the model was presented. However, in order to test that the model can be applied in the real world we need to make an extra analysis which is presented in the next chapter.

Chapter 4

Implementation of the Model

In the previous chapter we defined the theoretical monitoring model, but in order to prove it, we need to define a tangible object that allow us to test the feasibility of our model in the real world. In order to test it we will implement a tool. In this chapter we will define the conceptual architecture for the tool. Once the conceptual architecture is ready we will analyze its components, define a technological architecture and implement the tool according to the defined technological architecture.

4.1 Conceptual Architecture of the tool that proves the Model

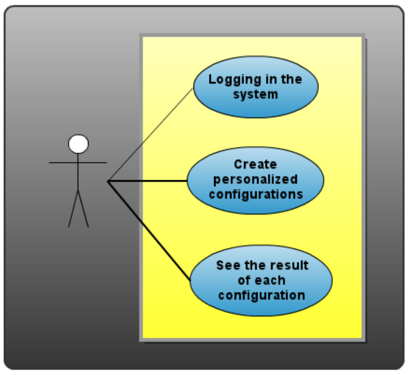

Logging in the system
Create personalized configurations
See the result of each configuration

Figure 4.1: Operations supported by the tool.

Before we define the conceptual architecture of the tool that will prove our model, is necessary to clarify that on the basis of the theoretical model presented in the previous chapter, we can already depict the most relevant operations that should be supported by the tool. It is really important to define these basic operations because they are considered in the design of our conceptual architecture, which is intended to be used as a reference to take the model from paper design to a possible real implementation. These operations are depicted in the figure 4.1.

Taking into account these basic operations in figure 4.1 and the General Business Process Monitoring Architecture presented in the section 3.1 of the previous chapter, it is clear that a monitoring system should consider three elements: Monitoring Client, Monitoring Infrastructure and the Business Process Engine.

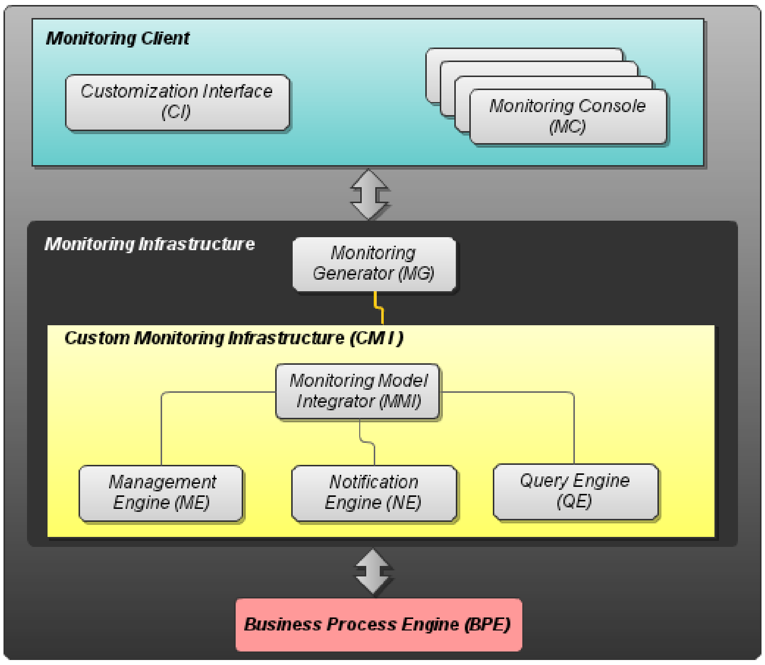

Monitoring Client
Customization Interface (CI)
Monitoring Console (MC)
Monitoring Infrastructure
Monitoring Generator (MG)
Custom Monitoring Infrastructure (CMI)
Monitoring Model Integrator (MMI)
Management Engine (ME)
Notification Engine (NE)
Query Engine (QE)
Business Process Engine (BPE)

Figure 4.2: Conceptual Architecture of the monitoring tool.

The conceptual architecture of the tool is shown in figure 4.2. Users access either the **Customization Interface (CI)** to specify their configurations or the **Monitoring Console (MC)** which will display the monitoring configurations that were defined previously. We need to point out that figure 4.2 shows several Monitoring Consoles because there should be a specific MC for each customer, and only one CI because all customers should use the same CI to create their configurations.

Figure 4.2 specifies also that Monitoring Client requests are always received by the **Monitor Generator (MG)**, which instantiates a **Custom Monitoring Infrastructure (CMI)** for each request. The CMI contains the **Monitoring Model Integrator (MMI)** which is the module in charge of coordinating the components required for both the extraction and management of the monitoring data from the **Business Process Engine (BPE)**, and the application of necessary options to satisfy user's customization requests. According to the dimensions of the monitoring model previously described, the CMI has three more components:

- **Query Engine (QE):** this executes the query required to extract the required monitoring data from the logging database of the **BPE**.
- **Management Engine (ME):** this implements the monitoring data policy specified by the user.
- **Notification Engine (NE):** this implements the monitoring data notification policy specified by the user.

As an example, the customization request of the situation described in the end of section 3.3 is instantiated by **MG** into the following business logic: Considering the V element, with $V = \langle p1, WaitingTime, \geq 2013/12/01, ALL, avg\rangle$. The MG instantiates the CMI where the MMI module makes QE able to run the following query on the BPE logging database:

```
select avg(wait_time)
from PROCESS, INSTANCE
where PROCESS.ID = INSTANCE.PROC and PROCESS.ID=p1 and INSTANCE.start > 2013/12/01
```

The MMI module requests ME to handle the M element, with $M = \langle persist, read \rangle$. This indicates that the information will be available in the future (because of the read option), and that historical information will be taken into account, so no matter if the information is pulled or pushed into the Monitoring Client, every update of information should be considered (because of the persist option). In this case, ME should have the business logic to handle not only the information, but also the communication with the Monitoring Client in order to indicate the Monitoring Console that the information will be available in the future and that historical information is considered.

Finally, the MMI module requests NE to handle the N element, with $N = \langle pull \rangle$. This indicates that the option chosen was "Information Pull", so NE should indicate the Monitoring Client how the interaction will be handled. Once the NE specifies this option to the Client, the Monitoring Console should provide the user a method so that he or she can request the information.

This example shows clearly that the model can map easily a real configuration where it is easy to see the parts that compose the configuration and how we can parse a specific request from the user into defined actions over the modules that are in the conceptual architecture. In the next section a more specific analysis of the components of the conceptual architecture is presented.

4.2 Practical Analysis of the Conceptual Architecture of the Tool

In this section a more detailed analysis of the conceptual architecture of the tool will be presented. The architecture involves three components Monitoring Client, Monitoring Infrastructure and Business Process Engine.

The Monitoring Client has two modules: **Customization Interface (CI)** and **Monitoring Console (MC)**. From a technical point of view these two components can be defined as an interface that allows the user to set a request and to see the results of the request.

The Monitoring Infrastructure is composed by several modules. One of them is the **Monitor Generator (MG)** which establishes direct communication with the MC. MG identifies the type of request and instantiates the **Custom Monitoring Infrastructure (CMI)**. The main component of the CMI is the module **Monitoring Model Integrator (MMI)**, which has the business logic to coordinate and manage the execution of the other three components: **Query Engine (QE)**, **Management Engine (ME)** and **Notification Engine (NE)**. When MMI finishes the execution of a request it manages the result and transfers it to the MG which has the logic to supply the MC and display the results to the final user.

According to the model defined in 3.3, **Management Engine (ME)** is a component that has the logic to provide the options of the sub-dimensions "Consumption" and "Storage". These options do not require an interaction with the BPE. Instead the operations performed by this module are related to the way the content is managed. Basically they determine only the time that the information will be available, and the option to provide historical data or not.

Notification Engine (NE) is other module that according to the model 3.3 has the logic to provide the options of the sub-dimension "Direction". It is also true that these options do not require an interaction with the BPE. Instead, the options of "pull" and "push" determine the way the information will flow between the monitoring infrastructure and the monitoring client.

Query Engine (QE) is an important module for the model because this component is taking care of establishing the connection between the CMI and the BPE. It is necessary to point out that the main objective of this chapter is to test the feasibility of our model, and that the success of the model strongly depends on the interaction between the CMI and the BPE, meaning that the success of the model depends on the idea that we can establish a connection between our CMI and the audit repository of a BPE. As it was explained in the previous chapter the connection is designed to be at a database interaction level. In order to verify this connection a brief research was performed and these are the results:

Pallas Athena FLOWer Since 1990s FLOWer have been implemented in many organizations for automating processes. Unlike other workflow management systems FLOWer is considered a case-driven system. Despite that this WfMS use a different paradigm, it is true that it also stores information related to process definition, case and work items, so we can see that we will find the monitoring variables that support the model [21].

IBM Websphere MQ Workflow According to the documentation [18] this WfMS uses the concept of "system". A system is a way in the system to define organizational units inside a company. The important point for us is that each system has a database. Besides, in a general view this WfMS has two main relational databases one to store process definition information and other to store information related to the transactions executed. The process definition information database stores data such as the list of activities that compose the process, its name, and status of the work items. Like the HPPM audit log that we analyzed in the literature study of this thesis, the database of IBM MQ Workflow also has an audit trail and each audit row that is stored registers the timestamp. With this notion, it is clear that even when we do not have a deep understanding of the IBM MQ Workflow architecture we know that the database has information related to the monitoring variables that we defined in our model in the previous chapter.

COSA COSA is a Petri-net based workflow which is mainly composed of both a server and a relational database [1]. Besides the database information, COSA has also an audit log that stores all the actions and errors. Just as the other systems did, COSA registers information about process definition, and case related information such as state and point in time.

TIBCO Staffware Process Suite Staffware is one of the most popular process management solutions in the industry since the 1990s. Staffware provides an audit log which register all the transactions at both case and process levels. In the information registered we find data such as process name, state and the timestamp of each transaction [17].

The previous list is the validation over the technical documentation of specific systems that the **Query Engine (QE)** module, which is defined to interact with a database of a BPE, will be able to find data related to the monitoring variables that are used by our monitoring model in most of the BPMS databases.

So far we have analyzed from a technical point of view all the components of our conceptual architecture and it is possible to say that the model seems feasible. In the next section a proposed technical architecture will be defined and finally a tangible tool will be implemented under the specification of the final technical architecture. This tool will help us to conclude if the model is really feasible or not.

4.3 From Conceptual to Technical Architecture

In 1980 the programming language Smalltalk, a pure object oriented language, showed a feature that became really popular up to the present day since important companies such as Sun Microsystems decided to use it for its toolkits and enterprise applications. This feature is the **Model/View/Controller (MVC) pattern**. This pattern has been really useful in the architecture of several information systems. MVC also known as Presentation/Abstraction/Control

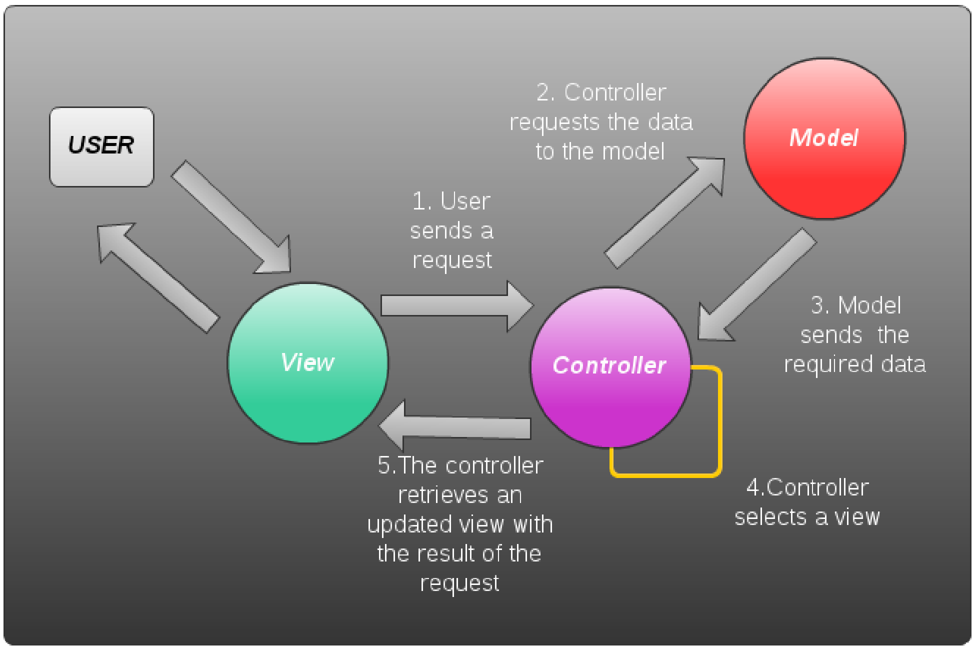

USER
View
Controller
Model
1. User sends a request
2. Controller requests the data to the model
3. Model sends the required data
4.Controller selects a view
5.The controller retrieves an updated view with the result of the request

Figure 4.3: MVC Logic Flow

design pattern has as its basic idea to separate the user interfaces from the data displayed by the user interface. The flow of the pattern is depicted in the figure 4.3.

There are three major components in **MVC** architecture:

- **Model** Element that encapsulates the "business logic" to manage the behaviour and data of the application.
- **View** Element that contains the representation of the data in the model. This element represents the presentation of the application and is not dependent on the application logic.
- **Controller** Element that communicates and coordinates the activities between the model and the view. This element basically intercepts requests from the view and passes them to the model to execute the appropriate action to fulfill the request.

The main objective of using a MVC architecture is the separation between both application data and business logic from the presentation layer to the user. The real benefits of using the MVC design pattern are:

- Reusability of the business logic.
- Flexibility to use several and different types of user interfaces.
- Specialization in the development (In terms of software development).
- Parallel development (In terms of software development).

Now with the clear definition of the MVC pattern it is easy to see that our conceptual architecture is composed by components which can be mapped to a MVC architecture. This integration is shown in figure 4.4 and is based on the following arguments:

- **Monitoring Client MC - View** The Monitoring Client is the interface with the final user. MI allows the user either to choose a configuration or to see the results of already defined configurations in the monitoring console module.

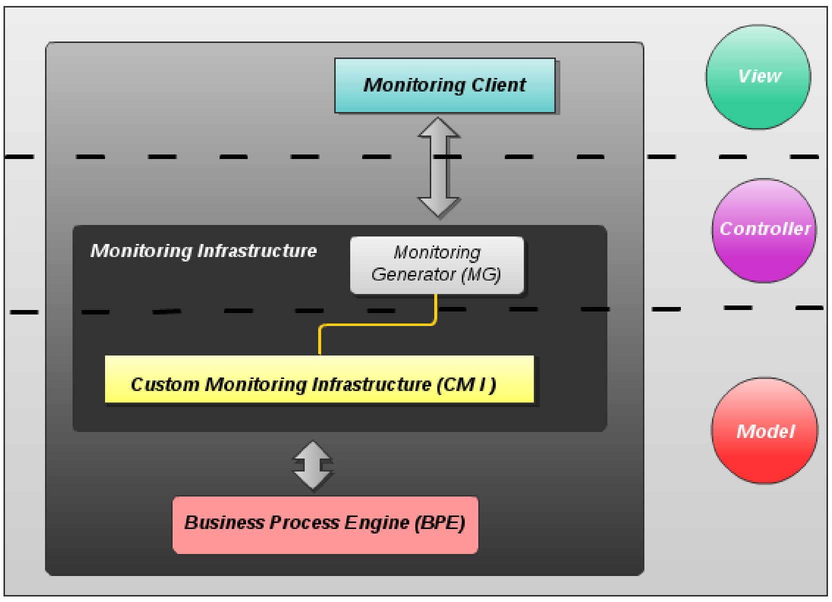

Monitoring Client
Monitoring Infrastructure
Monitoring Generator (MG)
Custom Monitoring Infrastructure (CM I)
Business Process Engine (BPE)
View
Controller
Model

Figure 4.4: MVC Applied to Conceptual Architecture

- **Custom Monitoring Infrastructure CMI - Model** The Custom Monitoring Infrastructure has the business logic to not only consider the content and information retrieval options, but also extract and manage the data from the BPE database.
- **Monitoring Generator MG - Controller** The Monitoring Generator is the connection point between the MC and the CMI.

In this section we analyzed the conceptual architecture and mapped to a real and proved software architecture such as the MVC. In the next section we will use one of the frameworks based on the MVC pattern to create the tool which will be used to test the model in a real business process execution environment.

4.4 Implementation of the Technical Architecture

In the previous section we defined that MVC architecture was a clear representation of the conceptual architecture of the tool that will prove our monitoring model. In real software development, developers use MVC frameworks that support the pattern and that reduce the development time. Some of the most popular MVC frameworks are:

- **Apache Tapestry** According to the documentation Tapestry is an open-source framework that allows the user to develop dynamic, robust, highly scalable web applications.
- **JSF** JSF is a framework based on the MVC pattern. Some of the disadvantages of using this framework are that there is no single source for implementation and that this framework presents a weakness related to security management when it is compared with other frameworks.
- **Spring MVC** This is a MVC framework which presents highly compatibility with UI technologies such as Tiles, PDF, JSP, etc. One of the disadvantages is that it exploits the flexibility of its components in such a level that the MVC gets modified and there is no common pattern controller.

- **Struts2** This is a MVC framework with a simple architecture, which is easy to extend. Its main characteristic is that is a Controller-based navigation. It has not only been in the market for more time than other frameworks, but also became the de-facto standard for web development for several years.

Considering the maturity of the framework, its documentation and popularity in the market, Struts2 was chosen as the framework to develop the tool to implement the model for this thesis project.

Struts2 is the next generation of Struts. Apache Struts was officially released in July 2001. Despite the fact that by the 2000s developers around the world did not have a clear framework for web development, they indeed had a clear idea that reusability and maintenance were concepts that were the basis for any tool or framework. On the basis of these aspects, Struts came in to scene and the number of web projects increased significantly since then. Some years later Struts2 was presented with the goal of making web development easier for the developers, for instance, one of the advantages of using a framework like this is the "encapsulation" since we have the ability to split an application into several modules. The use of modules allows developers to implement new features and prepare them to be independent. This is a critical characteristic that has been adopted by the architecture of Struts2. According to [28], there are different ways to split an application in modules. The following list depicts some cases that are or can be supported with Struts2:

- **Configuration information** An application can be split in different files. According to [28] this way makes the development easier because information is easier to find.
- **Application modules can be created as plug-ins** The idea is to group every component of a feature together so that it can be delivered as an independent plug-in.
- **Integration of new features as plugins** New features can be integrated to the application as plugins with minimal or no impact in the current performance of the system.

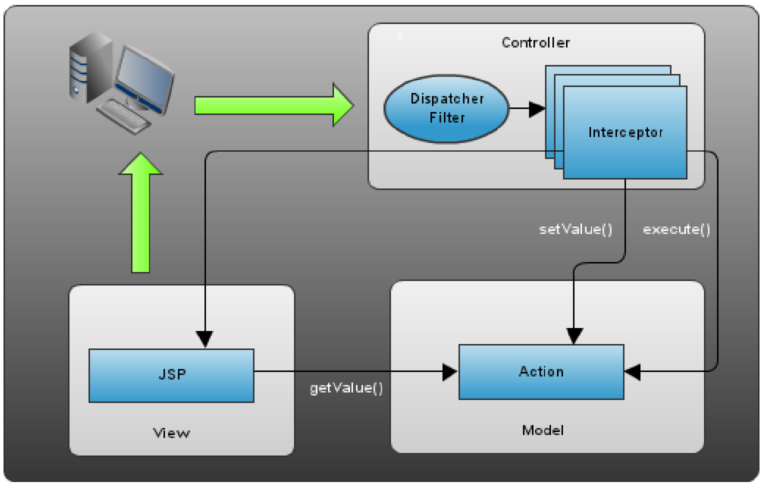


Figure 4.5: MVC in the Struts2 framework.

We have already stated that Struts2 is a MVC framework, however, we need to clarify that Struts2 is a pull-MVC also known as "MVC2". This means that Struts2 is based on the basic MVC pattern, but uses a different version from the traditional one. According to [28] the MVC pattern in Struts2 is composed by five core components: actions, interceptors, value stack, result types and view technologies. The controller is implemented with the Struts2 dispatch servlet filter and the interceptors, the model is implemented with actions, and the view is composed by the results from the action. Figure 4.5 displays the relationship of MVC and the Struts2 architecture.

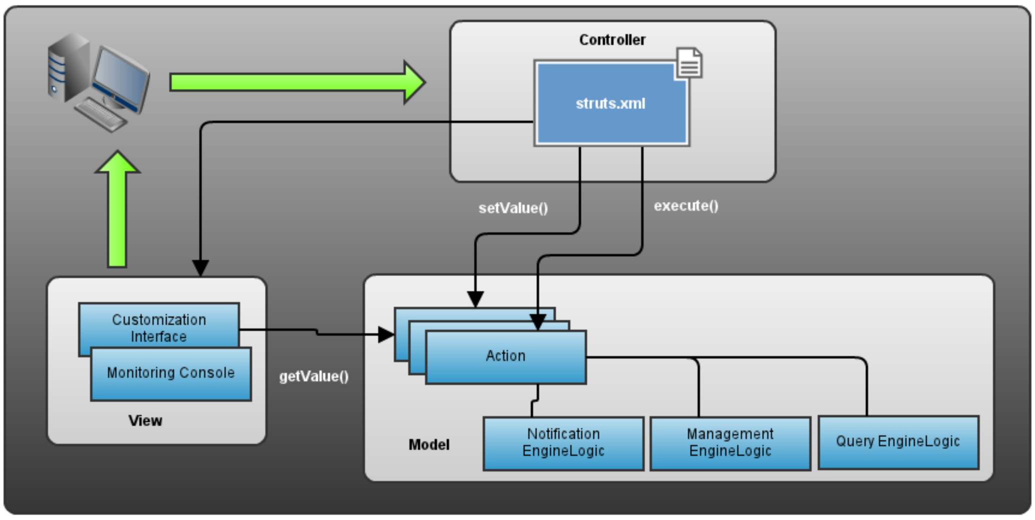


Figure 4.6: Mapping of the MVC architecture of the model into the MVC of Struts2.

Figure 4.6 maps the MVC architecture of the tool into the MVC of Struts. A description of mapping is explained below.

- Monitoring Client has two main components: Customization Interface(CI) and Monitoring Console(MC). Because we are using Struts2 then at least there will be two jsp files to represent these components.
- Monitoring Generator can be mapped to the Controller component of the Struts2 architecture, so in the final implementation there will not be a component with the name "Monitoring Generator". In this case the MG component will be the replaced by the "struts.xml", the core configuration file for the framework.
- Custom Monitoring Infrastructure (CMI) will be implemented with a set of action classes and other java classes with the business logic of the Management Engine, Notification Engine and Query Engine.

In this section we reviewed some of the MVC web based frameworks that are popular in the industry. Then we chose a framework and finally a high level mapping between the components of our technical architecture and the architecture of the chosen framework was presented. In the next section we will provide a deep analysis of the tool that was implemented in order to prove our model in a real environment.

4.5 Specifications of the Real Implementation

This section provides a deep analysis of the tool that was implemented in order to prove the proposed monitoring model of this thesis in a real environment.

4.5.1 BPEL

In the specific case of the implementation of the tool that will prove our model we decided to use a BPEL engine. Before we explain more details about the implemented tool, it is necessary to give a brief explanation about BPEL.

BPEL (Business Process Execution Language) came into scene as a standard of great potential for companies to meet their business goals and optimize their processes by integrating a variety of applications regardless of platforms and technologies associated with each activity. Besides, BPEL allows them to gain greater scalability and flexibility to cope with dynamism the business of today. Originally written by companies such as BEA Systems, IBM and Microsoft, with the support of OASIS (Organization for the Advancement of Structured Information Standards), BPEL is achieving strong support in the technology industry, especially in large companies, such as Oracle and Sun, as well Novell, Adobe and SAP, among others. Moreover, it enables companies to achieve high technological dynamism in its architecture, adapting quickly to change, whether internally or externally. In this way, they can more easily rearrange the communication between applications, significantly reducing the complexity of the processes.

The popularity of BPEL is based on the fact that it is an XML language that defines how a business process can be implemented using Web services. When a business process is implemented by Web services through BPEL it means that there will be a single interface to support XML messages associated platform-independent, thereby eliminating the need to use multiple protocols and formats and different interfaces. Even when not all the activities are currently implemented as Web services in organizations, the effects are tangible internally, since they help to simplify and make faster interaction and execution of a business process. With this notion, BPEL is sufficient to define any process flow with a logic, situation why it is considered as a key technology for the present and the near future of the industry.

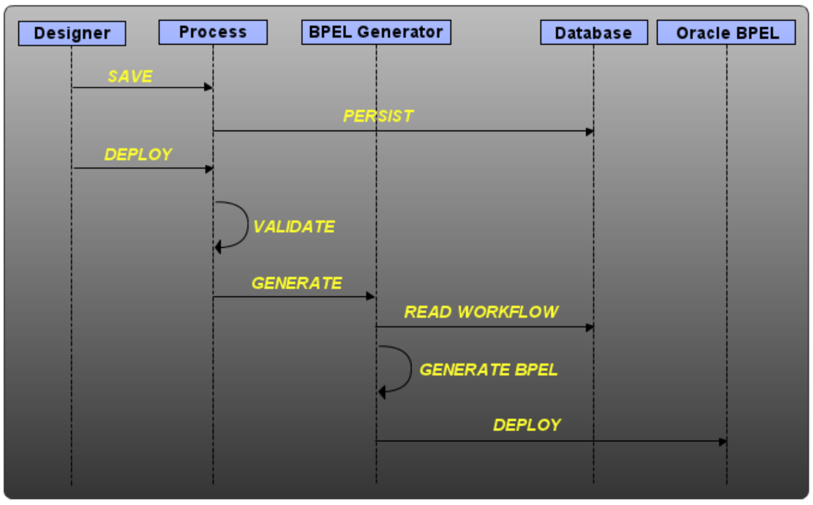


Figure 4.7: Building on-the-fly processes with BPEL.

Besides the integration advantages that BPEL provides, it gives the user the ability to build on-the-fly processes. According to the reference [27], the flow to create a process on the fly can be described in figure 4.7 and comprises the following steps:

1. Use of a designer to graphically model the business process.

2. The process definition is stored in the database by the designer

3. The BPEL Generator not only validates the process definition, but also generates both the BPEL XML file from the database representation and the associated files used for deployment.

4. The BPEL process is dynamically deployed on the BPEL Process Server.

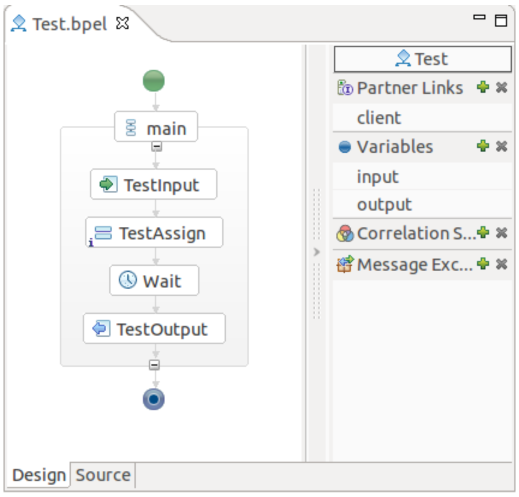


Figure 4.8: Building a process with Eclipse BPEL Designer Editor.

Both the popularity of BPEL in the market and the ability that it provides to build on-the-fly processes were two of the main aspects why we decided to use this technology. Now that we explained why BPEL was chosen we can start explaining how was the real implementation. As it is shown in the list of steps to build an on-the-fly process we can use a designer and we chose BPEL Designer Editor, an open source project under the Eclipse Technology Project. This Designer Editor provides support for integrating BPEL4WS design experience with the Eclipse platform. Figure 4.8 shows that we can easily create a business process with the editor.

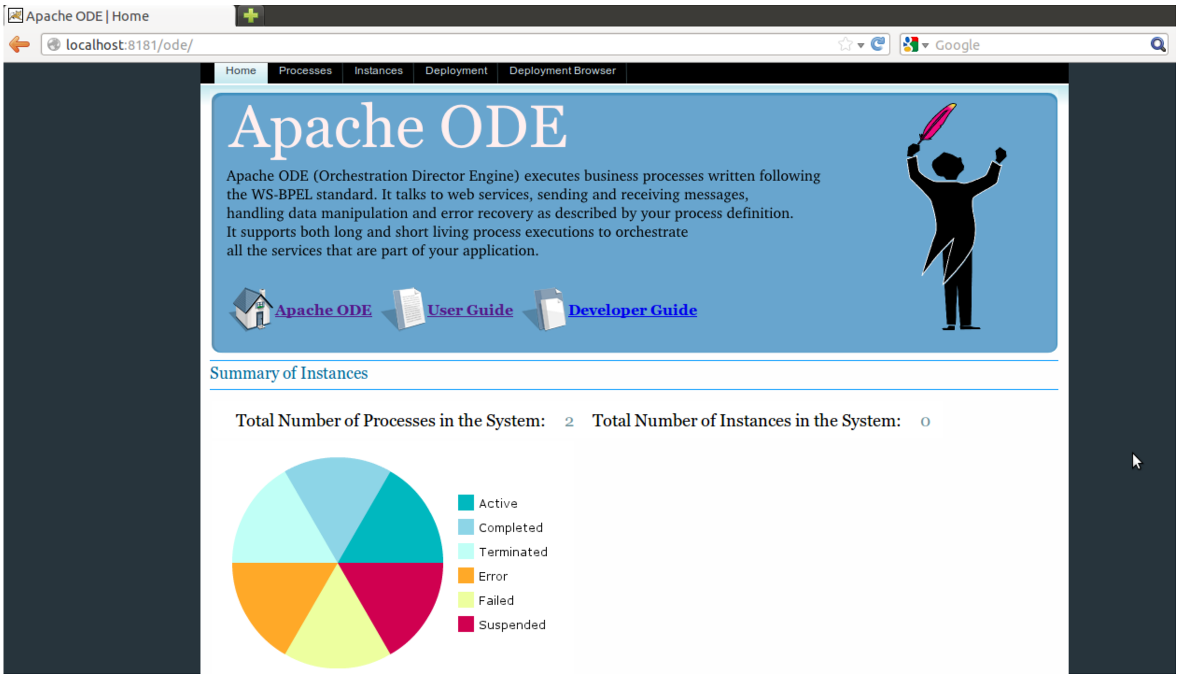
So far it is clear that we can create BPEL business processes with the Designer Editor, but these processes need an environment where they can be executed. For BPEL business processes, this environment is commonly known as Business Process Execution Runtime (BPER). A BPER has two main tasks to support. The former is to execute process logic, storing the processes' state. The latter is to handle the communication between processes and services. The BPER chosen for this thesis project is the orchestration engine of Apache. According to [11] "Apache ODE is an open-source BPER and uses Apaches Web Service engine Axis2 to realize the Web Service communication".

In the problem area of this thesis we stated that most of the process enactment technologies provide standard consoles for monitoring and the clear example is shown in figures 4.9, 4.10 and 4.11. These figures display the main monitoring information that can be found in the standard console provided by Apache ODE. Even when this console shows information about time and state, it is clear that it does not have neither the level of detail in the monitoring information, nor the level of customization that our model considers.

We have provided an idea of the relevant BPEL characteristics and why we chose it. Next section explain more details about the implemented tool.

4.5.2 Detailed Analysis of the Implementation

Now that we have a clear idea of the engine that we are using, we can explain with more detail how the tool was really implemented. Figure 4.12 depicts the architecture of the implemented tool, which we call "BPMonitor". It is clear that 4.12 is a more specific view of the architecture mapping presented in figure 4.6 of the previous section.

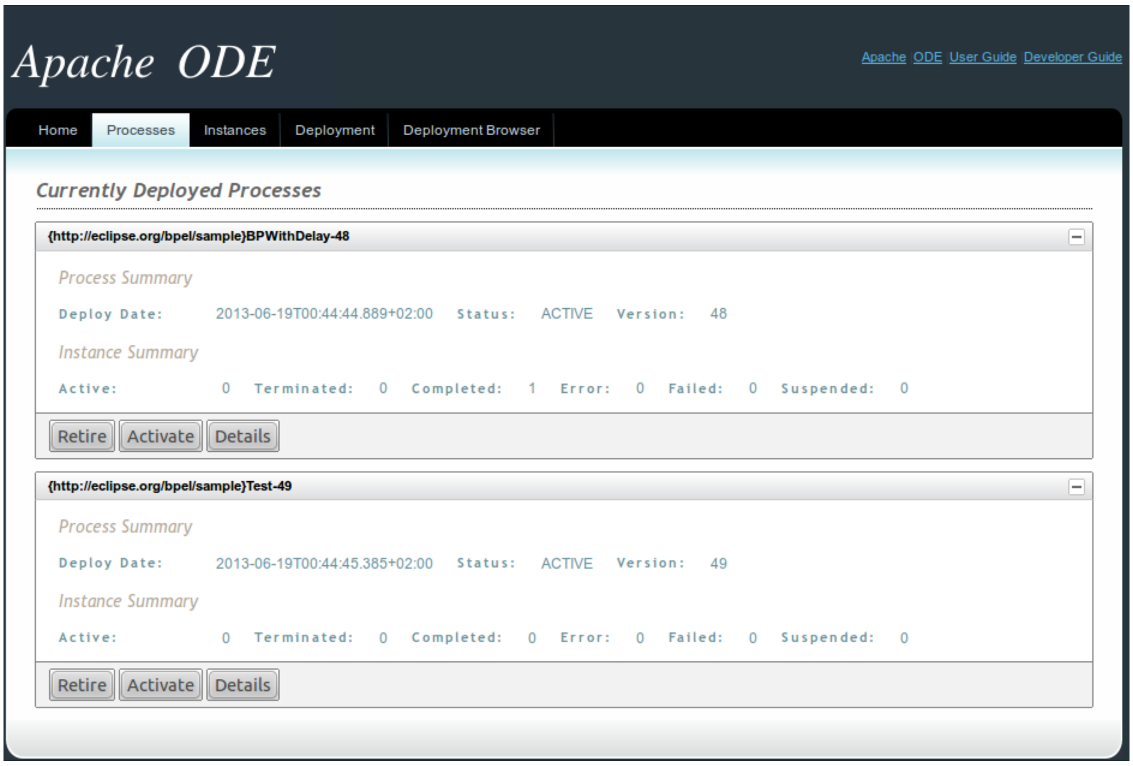


Apache ODE (Orchestration Director Engine) executes business processes written following the WS-BPEL standard. It talks to web services, sending and receiving messages, handling data manipulation and error recovery as described by your process definition. It supports both long and short living process executions to orchestrate all the services that are part of your application.

Summary of Instances

Total Number of Processes in the System: 2 Total Number of Instances in the System: 0

Figure 4.9: Apache ODE web console (home page).



Currently Deployed Processes

{http://eclipse.org/bpel/sample}BPWithDelay-48

Process Summary

Deploy Date: 2013-06-19T00:44:44.889+02:00 Status: ACTIVE Version: 48

Instance Summary

Active: 0 Terminated: 0 Completed: 1 Error: 0 Failed: 0 Suspended: 0

{http://eclipse.org/bpel/sample}Test-49

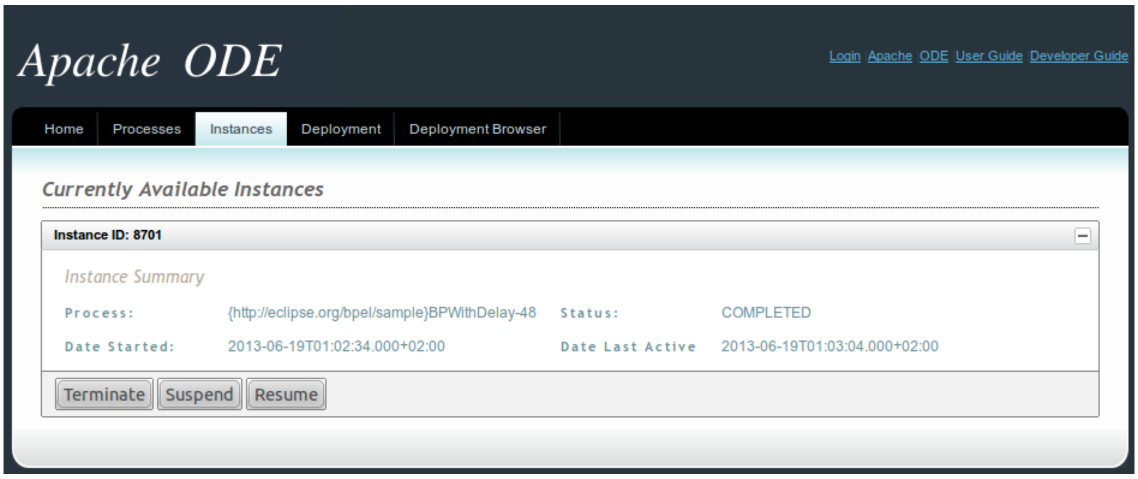
Process Summary

Deploy Date: 2013-06-19T00:44:45.385+02:00 Status: ACTIVE Version: 49

Instance Summary

Active: 0 Terminated: 0 Completed: 0 Error: 0 Failed: 0 Suspended: 0

Figure 4.10: Apache ODE web console (processes page).



Currently Available Instances

Instance ID: 8701

Instance Summary

Process: {http://eclipse.org/bpel/sample}BPWithDelay-48
Status: COMPLETED
Date Started: 2013-06-19T01:02:34.000+02:00
Date Last Active 2013-06-19T01:03:04.000+02:00

Figure 4.11: Apache ODE web console (instances page).

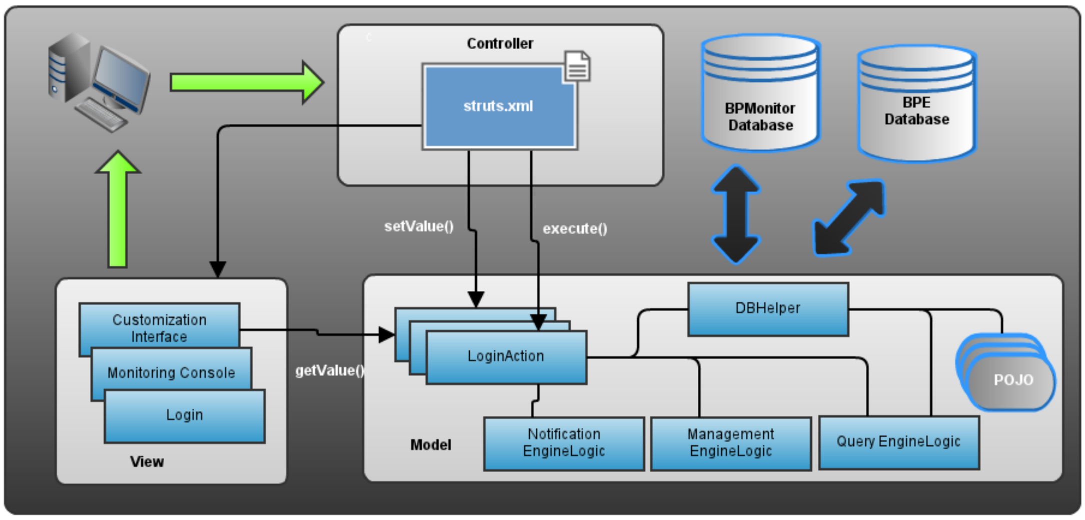


Figure 4.12: Detailed architecture of the implemented tool (BPMonitor).

Two elements that we find in figure 4.12 are databases. The former database is related to the BPEL engine and the latter to the internal database that supports the implemented tool. When we review the documentation of Apache ODE we find that ODE ensures the reliability of the processes execution by using a relational database to provide persistent storage. This database is composed by 24 tables, however, after an analysis and taking into consideration the monitoring variables that are used by the model we decided that we will use the tables depicted in figure 4.13. Figure 4.14 depicts the ER diagram of the internal database used by BPMonitor.

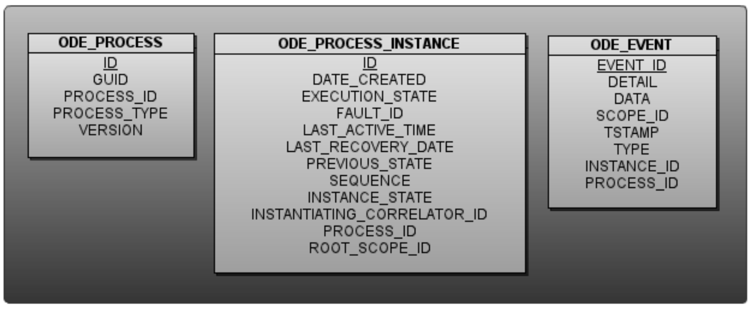


Figure 4.13: Apache ODE ER model (Tables of interest).

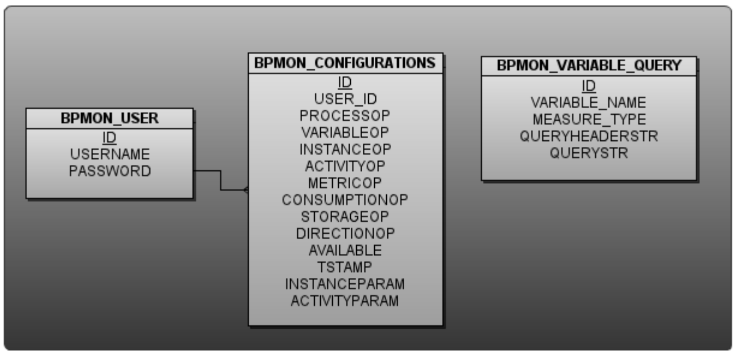


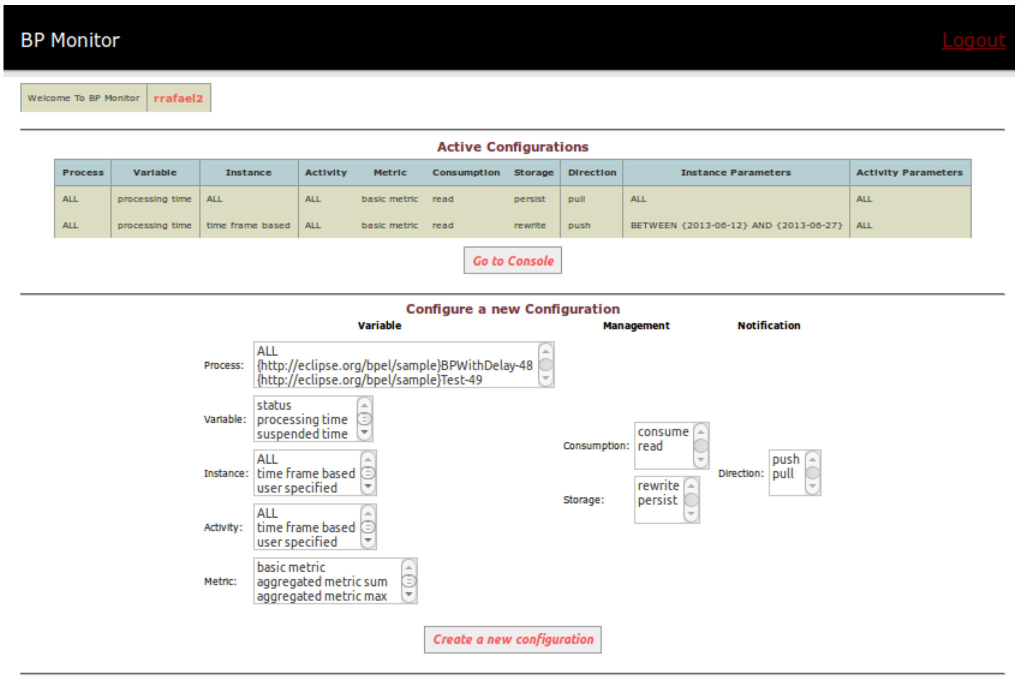
Figure 4.14: ER model of BPMonitor.

When we analyze the view component of the detailed architecture we see three main elements: Customization Interface, Monitoring Console, and Login. The Login interface is used to identify which user is accessing the tool. This interface is really important since the purpose of the tool is to provide customizable monitoring information not only to a single user, but to a set of different users, showing another advantage that our model has over the standard monitoring consoles.

Customization Interface is depicted in figure 4.15. This interface shows the configurations that the user has already created and allows him or her to create new customization specifications (We will use the term ”configuration” to make a reference to a specific customization specification). If we analyze the figure 4.15 we will see that it has clearly an interface where the user can find the multi-dimensional space of both monitoring dimensions and options that we defined in the previous chapter. This interface not only maps directly to the model defined in the previous chapter, but also allows the user to have the following flexibility when he is creating a configuration:

Process User can select one, a set or all the processes.

Variable User can select one or more monitoring variables at the same time.



Active Configurations

Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters
ALL	processing time	ALL	ALL	basic metric	read	persist	pull	ALL	ALL
ALL	processing time	time frame based	ALL	basic metric	read	rewrite	push	BETWEEN {2013-06-12} AND {2013-06-27}	ALL

Configure a new Configuration

Variable

Process: ALL / {http://eclipse.org/bpel/sample}BPWithDelay-48 / {http://eclipse.org/bpel/sample}Test-49

Variable: status / processing time / suspended time

Instance: ALL / time frame based / user specified

Activity: ALL / time frame based / user specified

Metric: basic metric / aggregated metric sum / aggregated metric max

Management

Consumption: consume / read

Storage: rewrite / persist

Notification

Direction: push / pull

Figure 4.15: BPMonitor - Customization Interface

Instance User can select a criterion to choose specific instances (all instances, user specified instances, or instances based on time)

Activity User can select a criterion to choose specific activities (all activities, user specified activities, or activities based on time)

Metric User can select a basic metric or one or more aggregated metrics at the same time.

Consumption User can select either "consume" or "read", but not both.

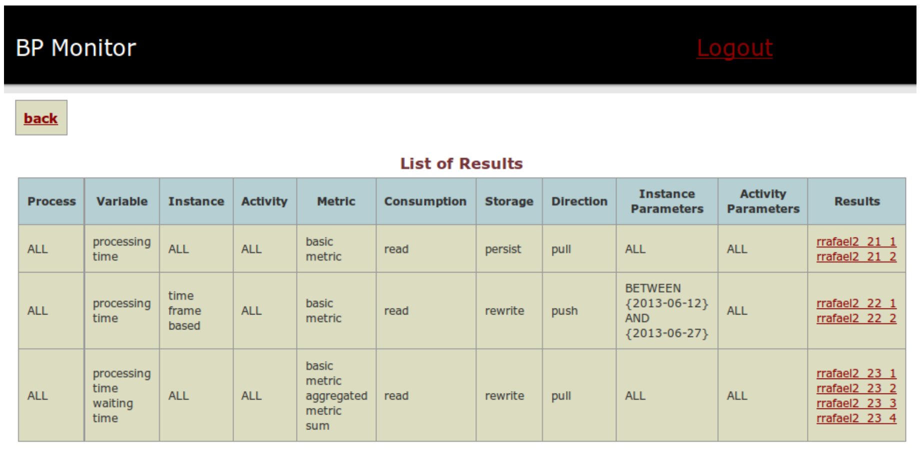
Storage User can select either "rewrite" or "persist", but not both.

Direction User can select either "push" or "pull", but not both.

It is necessary to clarify that every time when the user is logged in BPMonitor, he or she will always see the Customization Interface, and then the user can decide to create new configurations or just jump to the Monitoring Console and see the results for the active configurations.

The Monitoring Console is composed by two parts. The former is a list of the active configurations that the user defined. This interface is depicted in figure 4.16. The latter is the actual result of a configuration. Figure 4.17 is an example of a possible result interface to the user. When we analyze the figure 4.16 we can see that the table of has a column called "Results" with a set of links. When the user clicks on one of these links the second interface with the result will be displayed.

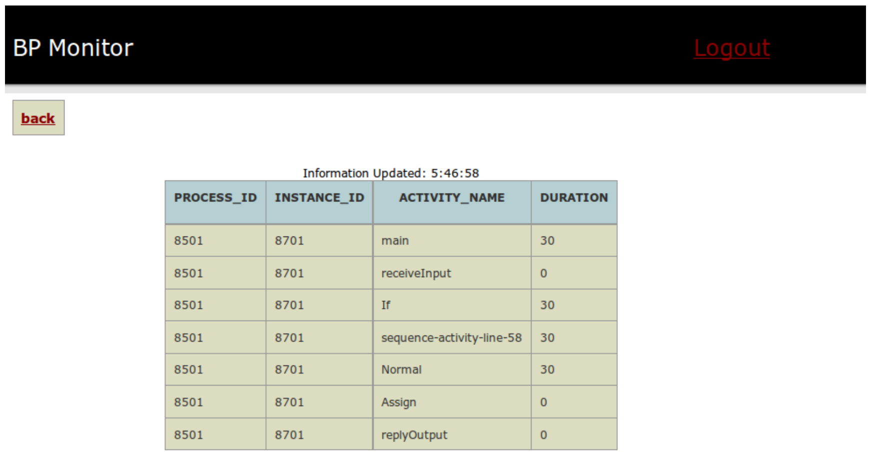
The controller is composed by the file "struts.xml" which is the core configuration file of Struts2. This file is an xml specification with the mapping between the view components and the actions of the model.



List of Results

Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters	Results
ALL	processing time	ALL	ALL	basic metric	read	persist	pull	ALL	ALL	rrafael2 21 1 rrafael2 21 2
ALL	processing time	time frame based	ALL	basic metric	read	rewrite	push	BETWEEN {2013-06-12} AND {2013-06-27}	ALL	rrafael2 22 1 rrafael2 22 2
ALL	processing time waiting time	ALL	ALL	basic metric aggregated metric sum	read	rewrite	pull	ALL	ALL	rrafael2 23 1 rrafael2 23 2 rrafael2 23 3 rrafael2 23 4

Figure 4.16: BPMonitor - Monitoring Console - List of Configurations



Information Updated: 5:46:58

PROCESS_ID	INSTANCE_ID	ACTIVITY_NAME	DURATION
8501	8701	main	30
8501	8701	receiveInput	0
8501	8701	If	30
8501	8701	sequence-activity-line-58	30
8501	8701	Normal	30
8501	8701	Assign	0
8501	8701	replyOutput	0

Figure 4.17: BPMonitor - Monitoring Console - Result of a specific Configuration

The model has several components such as Notification Engine Logic, Management Engine Logic and Query Engine Logic, however, in the final implementation the business logic of these was embedded in the body of the Struts2 actions, which are basically a set of java classes. Besides, we find a component DBHelper which is basically in charge of establishing communication between the monitoring tool and the databases that were previously described.

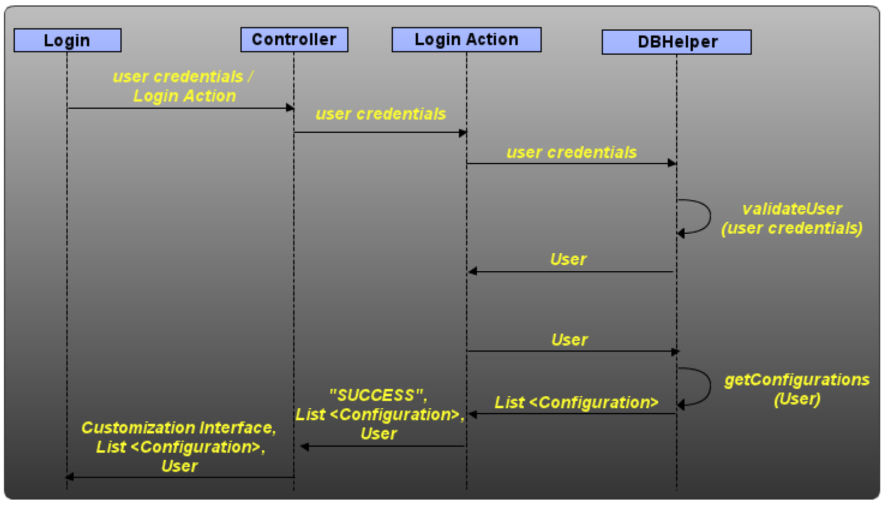

Login
Controller
Login Action
DBHelper
user credentials / Login Action
user credentials
user credentials
validateUser (user credentials)
User
User
getConfigurations (User)
List <Configuration>
"SUCCESS", List <Configuration>, User
Customization Interface, List <Configuration>, User

Figure 4.18: BPMonitor - Login Interface - Login Logic Flow

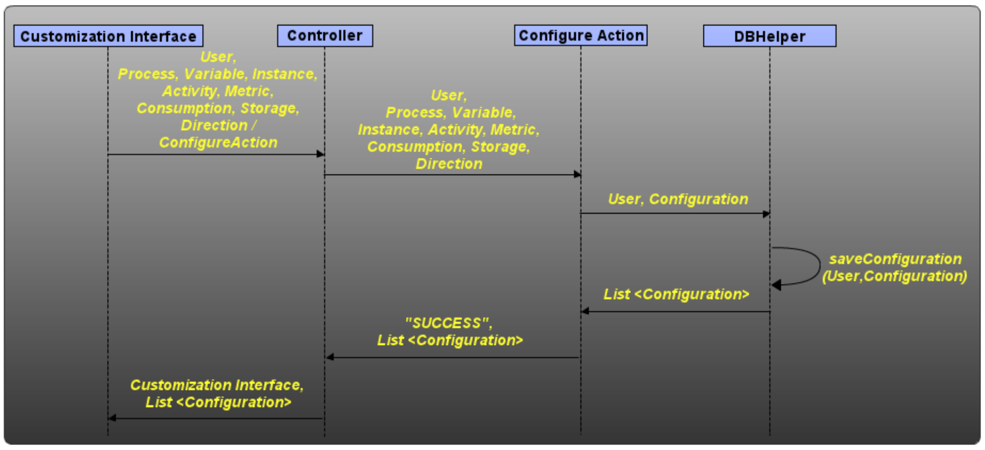

Customization Interface
Controller
Configure Action
DBHelper
User, Process, Variable, Instance, Activity, Metric, Consumption, Storage, Direction / ConfigureAction
User, Process, Variable, Instance, Activity, Metric, Consumption, Storage, Direction
User, Configuration
saveConfiguration (User,Configuration)
List <Configuration>
"SUCCESS", List <Configuration>
Customization Interface, List <Configuration>

Figure 4.19: BPMonitor - Customization Interface - Create Configuration Logic Flow

The flow of logic within the implemented tool (BPMonitor) is described in the following points:

Logging in the system The first step to use the monitoring tool is to provide the credentials of the user so that BPMonitor validates it and allows the access. If the user is a valid user for the system then the Login Action will request to the DBHelper instance the list of all the configurations created by the user in the past. It is important to say that Login Action has some of the logic of the engines (QE, ME, and NE) because this action checks the list of configurations and parses each of them in order to find the V, M, and N dimensions in all of them. Once each configuration is parsed, the Login Action will filter the configurations by taking away those that have a different value from "read" in the sub-dimension *Consumption* of the M dimension. When the list of configurations is filtered the Login Action will redirect

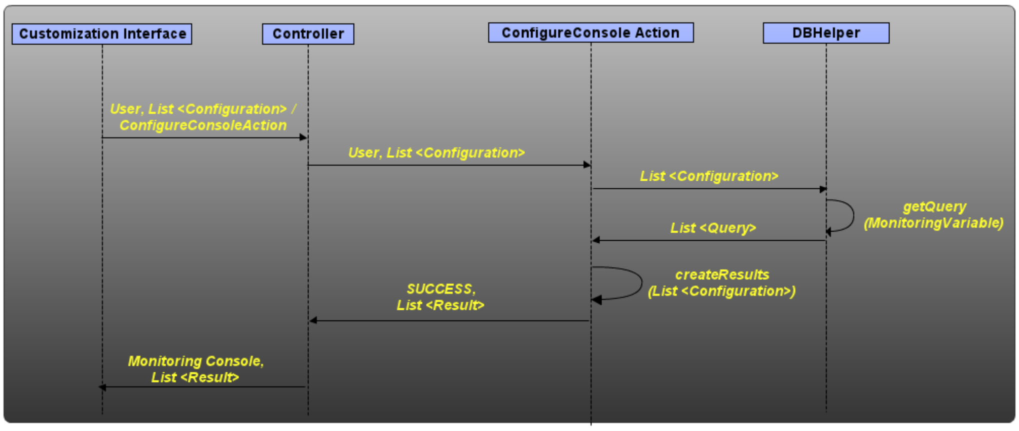

Customization Interface
Controller
ConfigureConsole Action
DBHelper
User, List <Configuration> / ConfigureConsoleAction
User, List <Configuration>
List <Configuration>
getQuery (MonitoringVariable)
List <Query>
createResults (List <Configuration>)
SUCCESS, List <Result>
Monitoring Console, List <Result>

Figure 4.20: BPMonitor - Customization Interface - Get Results Logic Flow

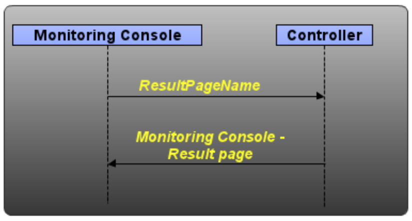


Figure 4.21: BPMonitor - Monitoring Console - Get Specific Result Logic Flow

the user to the Customization Interface, depicted in figure 4.15, which will show the filtered configurations to the user. This logic flow is depicted in figure 4.18.

Create a new Configuration Once the user has logged in the system he or she will see the Customization Interface, depicted in figure 4.15, where he or she can not only see the configurations that were already created, but also create new configurations. The logic starts when the user selects different values in the Customization Interface and when he decides to create a new configuration the action Configure Action will be instantiated. Configure Action will use the selected values to identify the dimensions:

$$V = \langle Process\ value, Variable\ value, Instance\ value, Activity\ value, Metric\ value\rangle$$

$$M = \langle Consumption\ value, Storage\ value\rangle$$

$$N = \langle Direction\ value\rangle$$

When the dimensions are identified the Configure Action can create a new instance of type Configuration, object that will be sent to the BPHelper in order to save this new configuration in the internal database of the tool. Once the new configuration is stored, DBHelper retrieves the updated list of configurations which will be updated in the Customization Interface, which will update the user about the status of the transaction. After this action, the user is free to repeat the process by creating more configurations or jump into the Monitoring Console to see the results for the current active configurations. The logic flow to create a new configuration is depicted in figure 4.19.

Jump into the Monitoring Console Once the user has logged in the system he or she will see the Customization Interface, depicted in figure 4.15, where he can jump into the Monitoring Console, depicted in figure 4.16, where the results for each configuration will be listed. This logic is depicted in figure 4.20. When the user decides to jump into the console, the ConfigureConsoleAction will loop in all the configurations, identifying the three main components

V, M, and N in each of them.

As it was explained in the previous chapter, the V component should be mapped into a SQL query, and this is the reason why in figure 4.20 ConfigureConsoleAction instance requests a list of SQL queries to the DBHelper instance, which is the class that communicates with the internal database in order to get the SQL queries that correspond to a specific monitoring variable. When ConfigureConsoleAction has the query for a specific monitoring variable it can apply the selected parameters such as the process, instances, and/or activities in order to have a SQL query that really considers all the values that are in V.

The M component which should be parsed by the ManagementEngine Logic should also be handled by the ConfigureConsoleAction instance, however, in the real implementation we have made some assumptions and one of them is to apply a value of "rewrite" to the Storage dimension in all the cases, which means that we will not consider historical information, so the tool only shows the information of the most recent request. With this statement clear, it is easy to see that there is no need to have logic of the ManagementEngine in the ConfigureConsoleAction.

The N component, which should be parsed by the NotificationEngine Logic, is also handled by the ConfigureConsoleAction instance which identifies and creates web interfaces, with the proper code to support the options information pull or information push depending on the value of N.

Open a specific result Once the user see the Monitoring Console, depicted in figure 4.16, he or she can see the list of configurations and each configuration has a set of links, each link is a reference to a specific result. When the user selects one of these results he will get a web interface similar to 4.17. The logic to open a specific result of a configuration is depicted in figure 4.21.

This section gave a detailed explanation of the final components and the real logic flow behind the implemented tool.

In this chapter we have given a deep explanation of how the theoretical model, presented in the previous chapter, can be taken from paper design to a real implementation. This process involved not only the definition and mapping from the conceptual architecture to a technical architecture, but also the mapping of the proposed technical architecture into a real software development framework, and the implementation of the framework. Next chapter will review some concepts that are important in order to evaluate the benefits that the model provides to users through the use of the tool.

Chapter 5

Evaluation of the Tool

In the previous chapter we gave a deep analysis of how the tool that proves our model was implemented. In this chapter we will address concepts such as software verification and validation in order to have a better evaluation of the tool, which is the physical representation of our model.

5.1 Verification and Validation

There are several definitions for these two terms, but in the field of software engineering verification is defined as ensuring that software conforms to its requirements, whereas validation is defined as ensuring that software meets the customer's needs.

According to what we previously stated, verification is closely related to the term "requirement", which is really important since according to the literature, system's utility and quality are determined by both the system's functionality and its non-functional characteristics such as performance and security. All of these characteristics are captured with requirements.

The functionality of the system is captured with "Functional Requirements (FR)". These type of requirements can be defined easily when we have a clear idea of what operations should be supported by the system, the work-flows performed by the system, the type of data that will be taken as input and the type of data that is expected as the output of the system.

On the other hand, the non-functional characteristics of a system can not be captured by Functional Requirements. According to Kotonya and Sommerville, there is a type of requirement called "Non-functional Requirements (NFR)" which represent "All requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet". Even when these are two separate types of requirements, they can be linked sometimes because we can have functional requirements that come from the analysis of the non-functional ones.

Summarizing, Functional Requirements represent what the system should do, and it is relatively easy to verify these requirements, but when we are dealing with non-functional requirements sometimes it is not as clear as we expect. Because of this situation, it is easier for us to review the existing models which classify these kind of requirements. Figure 5.1 depicts the non-functional requirements classification proposed by Sommerville. In our specific case, we are interested in the Product Requirements because they specify the characteristics of a system.

Even when the classification already establishes a set of Non-functional Requirements that are important, the reality is that the NF requirements that should be considered depend on the type

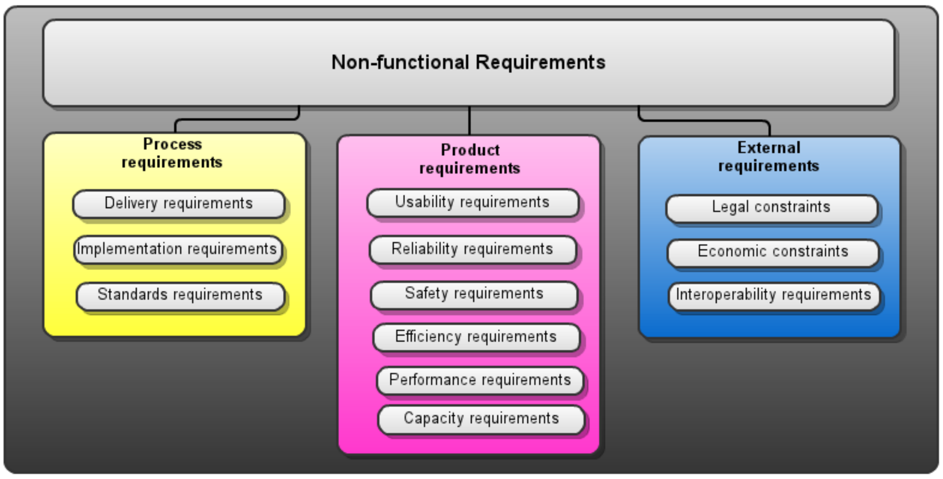


Figure 5.1: Sommerville Classification of Non-functional Requirements

of system that we have. In our case, because we are dealing with a tool that is intended to prove a customizable monitoring model, the tool should be able support not only customization, but also mass customization, situation that is already a parameter to choose relevant non-functional requirements to verify.

When we consider Mass customization, the term "Mass" indicates that the system should be able to support a great load of users, which indicates that the performance and scalability is a critical aspect for us. Besides, when we have a great number of users it is important to verify the usability of the system since it is important to assure that they can use the system as soon as possible. Finally, even when monitoring is not really a critical system, the fact that we have several users, implies that the system should be able to control the security of the configurations created by each user. Basically, *Performance requirements*, *Security requirements* and *Usability requirements* are relevant Non-functional Requirements for us.

5.1.1 Performance requirements verification

In order to explain how this aspect is addressed by BPMonitor, it is necessary to review in detail the logic flow to create the result pages and how they are presented to the user in the Monitoring Console. This flow is depicted in figure 5.2 and it is explained below:

1. As it was explained in the previous chapter when the user has logged in the application he will always see the Customization Interface where he decides to create more configurations or to jump into the Monitoring Console. When the user decides to jump into the Monitoring Console, the respective action class loops in the list of configurations of the user. For each configuration, this action class parses the configuration identifying the monitoring variable selected, and for each monitoring variable the action class will get a set of SQL queries that are stored in the internal database of the tool. Once the action class has all the queries, it takes the other values of a configuration in order to complete and have a SQL query ready for execution. Then, it creates dynamically a set of jps files and each of them will be in charge of executing a SQL query. In the end the action class stores the relationship of all the configurations and the result interfaces of each of them and this information is presented in the Monitoring Console, depicted in figure 4.16.

2. Once the user is in the the Monitoring Console, depicted in figure 4.16, he can choose to

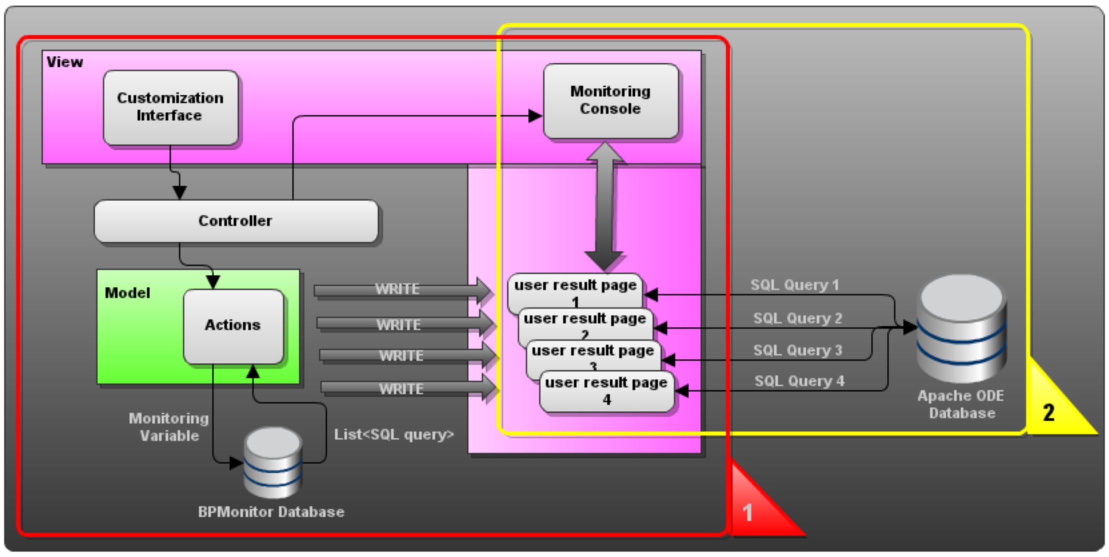


Figure 5.2: Logic Flow to create result interfaces.

open any of the result interfaces and then he will see a jsp similar to 4.17. It is important to explain that this jsp file does not have any complex logic, but instead only the necessary code to execute a specific SQL query that will be executed over the BPE database, which in this case is the database of Apache ODE.

It is important that we understand the previous logic flow because then it is possible to see that the scalability and performance of the tool strongly depends on the BPE database side. This approach has several advantages such as:

Use of simple logic in the interfaces that present the results to the user This implies that the tool will not have any problem with the use of memory, or the amount of data retrieved by the database, because the tool is not putting those values in memory, instead it is only displaying the query results.

Scalability and response time depend on the DMBS of the BPE database This is an advantage because even when our monitoring tool is running on a very basic server, it can still give fast service to the user because it depends on the DBMS. Most DBMS systems are client/server based and they usually operate over a powerful server, cluster of servers or in a storage area network environment.

Despite that this approach demonstrate advantages, we also need to consider that most of the BPE databases are operational databases, meaning that the data that is going into the database is used in real time to support the on-process activities of the BPE, and with our approach we are putting the load of monitoring activities in the BPE side, which may create an impact not only in the performance, but also in the availability of the BPE. Even when this is a great disadvantage for our monitoring model, the reality is that there are some ways to minimize the impact. When we analyze the core logic of our model we see that the V dimension is parsed into a SQL statement, which usually is a query statement that extract information from the database. With this notion, we find some strategies in order to improve the performance or at least avoid the impact that monitoring model could have in the operational processing of BPE. Some of these strategies are explained below:

Use of Indexes An index is a structure that provides faster access to data. Besides, one of the main advantages they provide is that in the case of relational databases, indexes can be

Manual Testing	**Automated Testing**
Manual tests are executed in sequence	Automated tests can be executed simultaneously on different machines
Manual tests are better suited for testing UIs	Automated tests are better suited for non-UI testing
Manual tests can be very time consuming	Automated tests can be cost-effective, when testers have to repeat tests numerous times

Table 5.1: Manual Testing vs Automated Testing

created, modified or even deleted without changing the database design or the application logic.

Use of Materialized Views A materialized view allows the user to create a summary of information, which not only avoids the need to perform several tasks in operational time, but also allows multiple queries looking for the same aggregation to reuse the summaries.

These are two of the most useful strategies that are considered when we are talking about performance tuning of the database aspect. So, in order to use our monitoring model and to minimize the impact in the operational performance of the BPE we should consider using at least one of these strategies.

So far we have described in detail the logic behind BPMonitor, its advantages and disadvantages, and the possible strategies to minimize the impact it may produce. However, we are still not sure about the real performance of our application and we are not even sure if the theoretical design can really support a great load of users. In order to find out this information we have resources such as ”Software Testing”. Despite that some software professionals hold strongly that testing costs too much, takes too much time, or does not help them build the product, we find also others who understand that testing is an investment in quality. Besides, it has been demonstrated that efficient testing produces a positive return, fitting within the overall project schedule and having quantifiable findings.

When we are talking about testing we find the term ”manual testing”. It is well known that this kind of testing is not only exhausting, but also inefficient and conflicts with the shorter application development cycles used in the recent days. Besides the time related issues, manual testing is prone to human error and may produce inconsistencies in the test results. In order to overcome some of the disadvantages of manual testing we find the term of ”test automation”. We need to clarify that test automation may not test better than the human tester, but if it is implemented correctly, it can certainly be faster, helping testers to test better and more effectively. Table 5.1 shows a comparison between automated and manual testing.

As it is described in table 5.1, automated testing helps to reduce testing time and it has some benefits when we need to repeat tests. These two aspects are important for us since we are interested to know the performance of our tool. This kind of testing is known as ”Performance testing” and can be performed in an automated way.

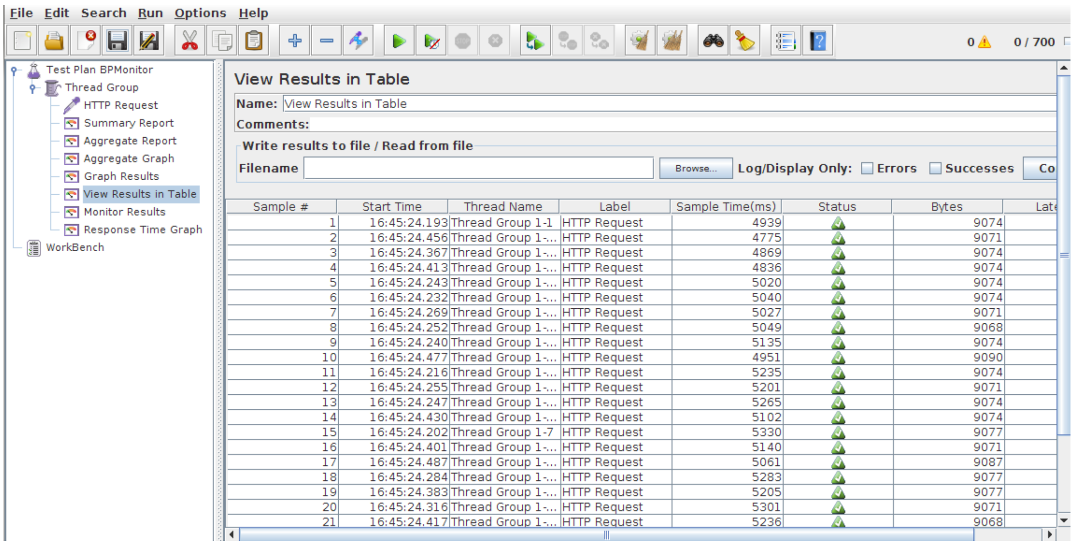
In the real world, there are several test automation tools or softwares available, and their cost may vary from zero to tens of thousands of dollars. Our choice will depend on our needs, available resources, and budget of the testing project. In the specific case of our project, we need a tool that requires the minimum acquisition cost, has a low learning curve, allows users to test web applications, and has good support and documentation. Taking into account the previous arguments we decided to use Apache JMeter.

Users	Samples	AVG(ms)	MIN(ms)	MAX(ms)	Rate	AGT Bandwidth	AVG Bytes
10	20	48	42	56	20.10	178.1	9073.1
20	40	55	42	81	37.84	335.2	9072.0
50	100	1202	46	1687	28.64	253.7	9072.3
100	200	2664	315	3333	31.75	281.3	9072.6
200	400	5651	3730	6450	32.38	287.0	9077.5
500	1000	10985	1453	28011	29.48	261.4	9079.2
700	1400	14119	626	47677	27.45	243.5	9080.2

Table 5.2: BPMonitor Stress Test Results

According to [13], JMeter is a software designed to test performance and functional behavior of client/server applications, such as web applications. JMeter is one of the most widely used open source testing applications that are in the market. It is a Java-based application, situation that makes it highly extensible through a provided API. JMeter takes the role of the ”client” in a ”client/server” application. Basically, it measures response times, CPU loads, memory usage, and resource usage. Although it was originally designed for testing web applications, it has been extended to support other test functions. JMeter is being developed under the Apache Jakarta Projects.

In the reference [13] it is stated that ”JMeter being a highly robust, scalable, and portable application makes it a suitable testing tool for today’s non-proprietary, fast-changing, and market-driven application development process. Anyone with software testing experience or knowledge at any level will find JMeter easy to learn and use.”. Now that we have a clear idea of what is JMeter and what can we do with this tool we will explain how we use it in order to test the performance of BPMonitor.



View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Late
1	16:45:24.193	Thread Group 1-1	HTTP Request	4939	✓	9074	
2	16:45:24.456	Thread Group 1-...	HTTP Request	4775	✓	9071	
3	16:45:24.367	Thread Group 1-...	HTTP Request	4869	✓	9074	
4	16:45:24.413	Thread Group 1-...	HTTP Request	4836	✓	9074	
5	16:45:24.243	Thread Group 1-...	HTTP Request	5020	✓	9074	
6	16:45:24.232	Thread Group 1-...	HTTP Request	5040	✓	9074	
7	16:45:24.269	Thread Group 1-...	HTTP Request	5027	✓	9071	
8	16:45:24.252	Thread Group 1-...	HTTP Request	5049	✓	9068	
9	16:45:24.240	Thread Group 1-...	HTTP Request	5135	✓	9074	
10	16:45:24.477	Thread Group 1-...	HTTP Request	4951	✓	9090	
11	16:45:24.216	Thread Group 1-...	HTTP Request	5235	✓	9074	
12	16:45:24.255	Thread Group 1-...	HTTP Request	5201	✓	9071	
13	16:45:24.247	Thread Group 1-...	HTTP Request	5265	✓	9074	
14	16:45:24.430	Thread Group 1-...	HTTP Request	5102	✓	9074	
15	16:45:24.202	Thread Group 1-7	HTTP Request	5330	✓	9077	
16	16:45:24.401	Thread Group 1-...	HTTP Request	5140	✓	9071	
17	16:45:24.487	Thread Group 1-...	HTTP Request	5061	✓	9087	
18	16:45:24.284	Thread Group 1-...	HTTP Request	5283	✓	9077	
19	16:45:24.383	Thread Group 1-...	HTTP Request	5205	✓	9077	
20	16:45:24.316	Thread Group 1-...	HTTP Request	5301	✓	9071	
21	16:45:24.417	Thread Group 1-...	HTTP Request	5236	✓	9068	

Figure 5.3: Stress Testing over BPMonitor using JMeter

Once we have downloaded and installed JMeter, we only need to define a Test Plan which normally includes a Thread group and a set of Listeners. These two elements are explained below:

- A Thread group is an element that is used to specify number of running threads, a ramp-up period, and loop count. Each thread simulates a user, the ramp-up period specifies the time

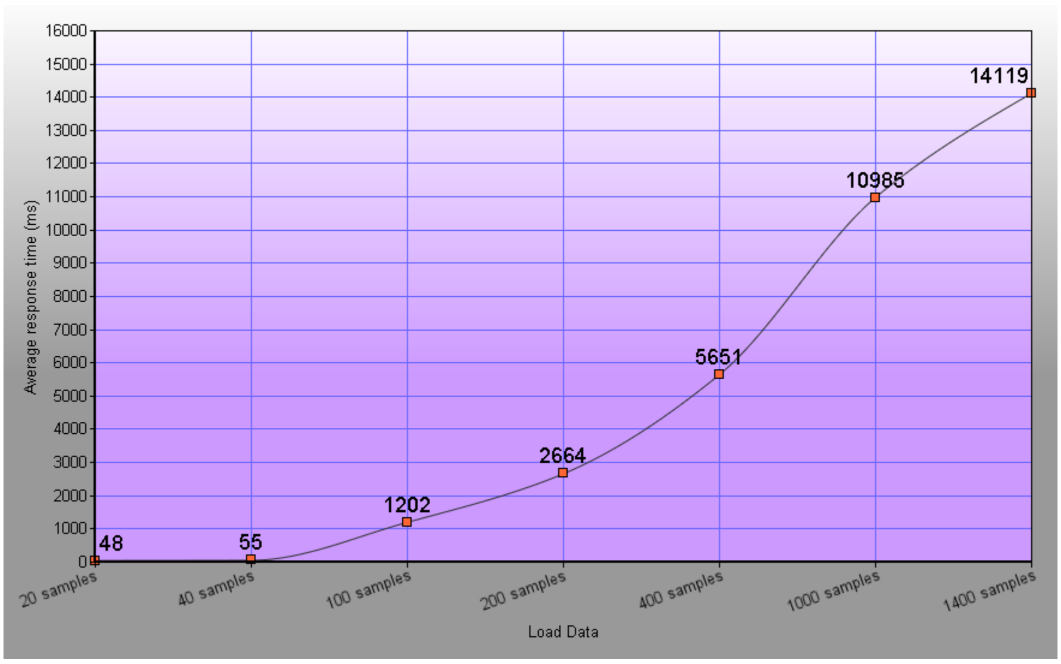


Figure 5.4: Stress Test - BPMonitor (Monitoring Console) Response time results.

to create all the threads and the loop count defines the number of times the test will be repeated for the thread group.

- A Listener is used to process the result of the requested data. This component help us to see the results of a test.

In our case, we are interested to know not only if BPMonitor is really able to handle a great load of users, but also what is its performance in a real stress testing situation. With this notion, we decided to perform a stress test over BPMonitor using JMeter. In order to complete this, first we need to define the several test plans that will be executed and the listeners that we can use from JMeter.

Once we defined both the test plans (specifying number of users, loop counts and ramp-up periods) and the listeners, we decided to configure them in JMeter and start the stress tests where JMeter literally simulated several users requesting their specific Monitoring Console from our BP-Monitor tool. Figure 5.3 depicts the moment when JMeter is executing one of the test plans pointing to BPMonitor, sending http requests to get different Monitoring Consoles.

The obtained results of our performance tests are displayed in table 5.2. This table indicates the metrics measured for the several test plans, each test plan has a different number of users, but each of them has a ramp-up period of 1 second and a loop count of 2. In order to understand the results of table 5.2 we can review any row from the table. Let us consider the last row where we find the results of a test plan configured with 700 users (using a ramp-up period of 1 second), and a loop count of 2 which really represents a total of 1400 requests to the application that is being tested. This is the reason that explains the 1400 that is displayed in the number of samples of this row. Besides, the table with the results indicates data about maximum, minimum and average response times of BPMonitor(In this specific case we are talking about the time it takes to the Monitoring Consoles to retrieve monitoring data), the aggregated bandwidth registered in each test execution and the average number of bytes registered.

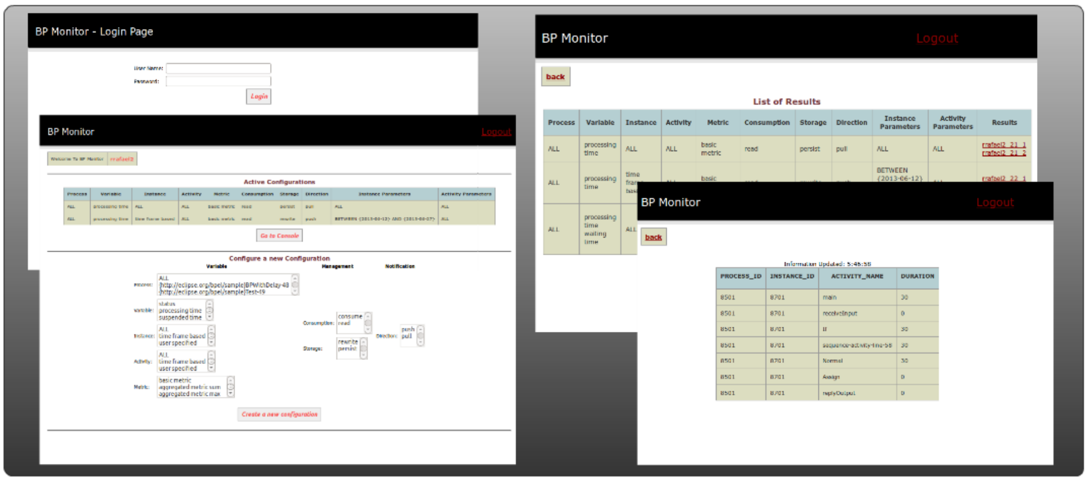
It is necessary to state that during the stress tests we also checked if the time to create a new configuration in BPMonitor was affected by the load of users in the system and we confirmed that this process does not suffer any impact. Let us consider an example where 50 users are opening their consoles and user 51 decides to create a new configuration, the reality is that user 51 will not be affected since getting monitoring data and creating a new configuration are working on separate databases.

The results of the stress test help us to confirm that BPMonitor will not have any problem related to memory because the number of bytes remains constant in all the cases. Besides, the results show that even when BPMonitor demonstrate an increment in the response time when the load of users increases, as it is depicted in figure 5.4, BPMonitor remains available even when it receives a load of 1400 requests with an approximate rate of 30 per minute.

After making the theoretical analysis over the logic behind BPMonitor and the real performance testing we are sure that BPMonitor is ready to support a great load of users, so it confirms that its design can support a "mass" situation, and even when the the response times increased with the load of requests, it still was available accepting requests. It is important to state that the results of the test strongly depend on the hardware constraints that are present where the application is installed. It is true that the performance results would be much better in a real scenario where we have a server dedicated for BPMonitor and a server dedicated to the BPE.

So far we presented a clear overview of how the design of BPMonitor supports a great load of users. Besides, we also present the results of a real stress test performed over the monitoring console of BPMonitor. In the next section we will address another non-functional requirement that the system should consider.

5.1.2 Usability requirements verification



Active Configurations

Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters
ALL	processing time	ALL	ALL	basic metric	read	persist	pull	ALL	ALL
ALL	processing time	time frame based	ALL	basic metric	read	rewrite	push	BETWEEN {2013-06-12} AND {2013-06-27}	ALL

List of Results

Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters	Results
ALL	processing time	ALL	ALL	basic metric	read	persist	pull	ALL	ALL	rrafael2_21_1 rrafael2_21_2
ALL	processing time	time frame based	[illegible]	basic	[illegible]	[illegible]	[illegible]	BETWEEN {2013-06-12}	[illegible]	rrafael2_22_1
ALL	processing time waiting time	ALL								

Information Updated: 5:46:58

PROCESS_ID	INSTANCE_ID	ACTIVITY_NAME	DURATION
8501	8701	main	30
8501	8701	receiveInput	0
8501	8701	If	30
8501	8701	sequence-activity-line-58	30
8501	8701	Normal	30
8501	8701	Assign	0
8501	8701	replyOutput	0

Figure 5.5: BPMonitor View interfaces.

A usability requirement can be clearly defined as "how easy the system must be to use". In the paper [23] it is stated that usability requirements "must be tangible in order to verify them and trace them during the development, and complete so that if we fulfill them, we are sure that we get the usability we intend". This statement represents two main challenges for us, however, we can check the traditional definition and find a way to analyze if our system fulfills them. According to [23], the traditional definition of usability consists of five factors:

Ease of learning The system must be easy to learn for both novices and users with experience from similar systems.

Task efficiency The system must be efficient for the frequent user.

Ease of remembering The system must be easy to remember for the casual user.

Understandability The user must understand what the system does.

Subjective satisfaction The user must feel satisfied with the system.

Figure 5.5 depicts the Login interface, Customization interface and the two interfaces of the Monitoring Console. It is clear to see that these are the 4 main components of the tool.

When we analyze these interfaces we can see that even when they do not have any complex visualization effect, they are indeed simple interfaces, easy to understand, to learn and to use. Besides, the majority of the interfaces use tables to present the information, and even when in the recent days people are used to see rosters, and charts, it is still true that tables are still present not only in professional life, but in domestic activities as well. According to the literature, a table is a meaningful way of organizing statements and figures so that the audience can quickly identify and understand how they relate with each other. Besides, the main advantage of tables is that they can fit a great amount of information into a simple and well organized structure.

So far we presented an overview of how the design of BPMonitor supports the usability aspect. In the next section we will address another non-functional requirement that the system should consider.

5.1.3 Security requirements verification

According to [4] "Security Requirements are caused to exist because people and agents that they create (such as computer viruses) pose real threats to systems in the world. Security is thus unlike all other areas in a specification as someone is consciously and deliberately trying to break the system". Besides, literature states that security requirements are based on the following goals:

- Ensure integrity of the system.
- Ensure controlled access to the system and to the information provided.

In the recent years one of the main challenges brought by technology was to have an effective management of authorization to data. In fact, one of the first strategies to ensure security is the use of access control lists (ACLs). Server administrators started to use ACLs in order to control user access to documents inside organizations. At the beginning, ACLs were a great improvement in the security aspect, but they were not completely flexible since they were linked to particular objects and these objects were tied to a particular user. This situation implied that we had a lot of users and each user linked to many permissions, having a large number of user-permission associations to manage. Because of these arguments, some time later ACLs were replaced by Role-Based Access Control (RBAC). This strategy involved the use roles, where permissions were authorized for roles, and roles were associated to users. In reality the implemented tool only supports validation of users. This means that the current state of BPMonitor is working under a very basic ACL schema where it is necessary to register the users that can use the system and then the application will validate if the provided credentials are valid in order to allow access to the system. Even when in possible future work the database of BPMonitor could support RBAC, so far we can say that this basic approach is already giving us a simple method to ensure controlled access to the system and to the information provided

The second goal of security requirements is related to integrity and when we are talking about integrity of the system we need to consider techniques that can restrict the level of privacy of both data and messages, situation that is closely related to the security of the communication channels. Indeed, it is well known that integrity is achieved by combining strategies such as cryptography, hash functions and/or digital certificates. A digital certificate is an electronic document based on both public key cryptography and a digital signature system. Its primary mission is to ensure the relationship between a person, entity and/or a web server. Digital Certificates usually link a person or entity with a public key, and by the use of a digital signature, the system ensures that the certificate you receive is really from the person that claimed to sent it. From the point of view of the functionality, electronic certificates can be divided into:

SSL Certificates (Client side)
These certificates are used to identify and authenticate client to server communications using the Secure Socket Layer, and is usually issued to a person.

SSL Certificates (Server side)
These certificates are used to identify a server to client in communications using the Secure Socket Layer, and are generally delivered on behalf of the company owning the secure server.

MIME Certificates
These certificates are used for signed and encrypted email, which is usually issued to an individual.

Certifying Authority Certificates
These certificates identify recognized Certifying Authorities. Clients use these certificates in order to determine the trust of other certificates.

Because of the nature of BPMonitor and the type of integrity we are looking for, we need to consider those certificates related to SSL or Secure Sockets Layer simply because we are intended to establish communication over a secured connection. If the communication channel is secure we are already ensuring part of the integrity of the system, but when we use certificates then we are proving that the communication is being performed by the right partners. Particularly, it is important to state that a SSL Certificate ensures online visitors that confidential information and transactions can not be seen, or intercept or altered by an unauthorized third party when transmitted.

So far it is clear that SSL is important for us, but we have not explained how we are going to adapt SSL to the implemented tool. In our case, BPMonitor is a Struts2 web application which requires to be deployed in a servlet container. We chose to use Tomcat, which is a free servlet container where servlet and JSP codes can be tested. According to Sun Mycrosystems Tomcat is a fully J2EE compliant servlet container. In order to setting up SSL on Tomcat we need to complete the following steps:

Creating a keystore file Use the java/keytool command to create the keystore file.

Configuring Tomcat to use the keystore Make the proper changes in the server.xml file in order to enable SSL port of Tomcat server.

Testing if SSL is available in Tomcat We will try to access the web console of Tomcat using the port of SSL.

Accessing BPMonitor with the SSL port We have to open BPMonitor using the port of SSL

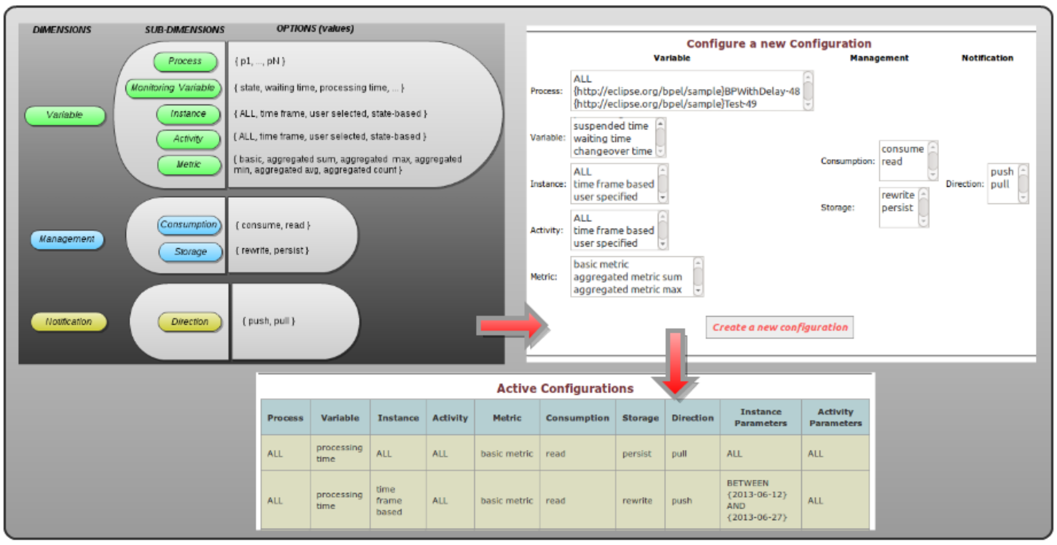
So far we have analyzed not only the strategies of security that are applied directly in BP-Monitor, but also those modifications such as the SSL setting changes in Tomcat server, that are required in order to achieve the security requirements goals. In the next section we will address briefly the functional requirements and we will check the software validation aspect.

5.1.4 Functional requirements verification and validation of the system

At the beginning of the previous chapter, when we defined the conceptual architecture, we already presented in figure 4.1 a UML case diagram with the three high level functional requirements that should be considered by our application. Even when the figure calls them "Operations supported by the tool", the reality is that they are the basic functional requirements that should be present in the final application.

Both the basic functional requirements and the respective components of BPMonitor that fulfill them are described below:

- Logging in the system - BPMonitor has a Login interface and a set of action classes that handles user credentials and provides access control.
- Create personalized configurations - BPMonitor has a Customization Interface and a set of action classes that handles the creation of configurations.
- See the result of each configuration - BPMonitor has Monitoring Console Interface and a set of other jsp interfaces that handles the display of results to the user.


DIMENSIONS
SUB-DIMENSIONS
OPTIONS (values)
Variable
Process
{ p1, ..., pN }
Monitoring Variable
{ state, waiting time, processing time, ... }
Instance
{ ALL, time frame, user selected, state-based }
Activity
{ ALL, time frame, user selected, state-based }
Metric
{ basic, aggregated sum, aggregated max, aggregated min, aggregated avg, aggregated count }
Management
Consumption
{ consume, read }
Storage
{ rewrite, persist }
Notification
Direction
{ push, pull }
Create a new configuration

Configure a new Configuration

Variable — Management — Notification

Process: ALL / {http://eclipse.org/bpel/sample}BPWithDelay-48 / {http://eclipse.org/bpel/sample}Test-49

Variable: suspended time / waiting time / changeover time

Instance: ALL / time frame based / user specified

Activity: ALL / time frame based / user specified

Metric: basic metric / aggregated metric sum / aggregated metric max

Consumption: consume / read

Storage: rewrite / persist

Direction: push / pull

Active Configurations

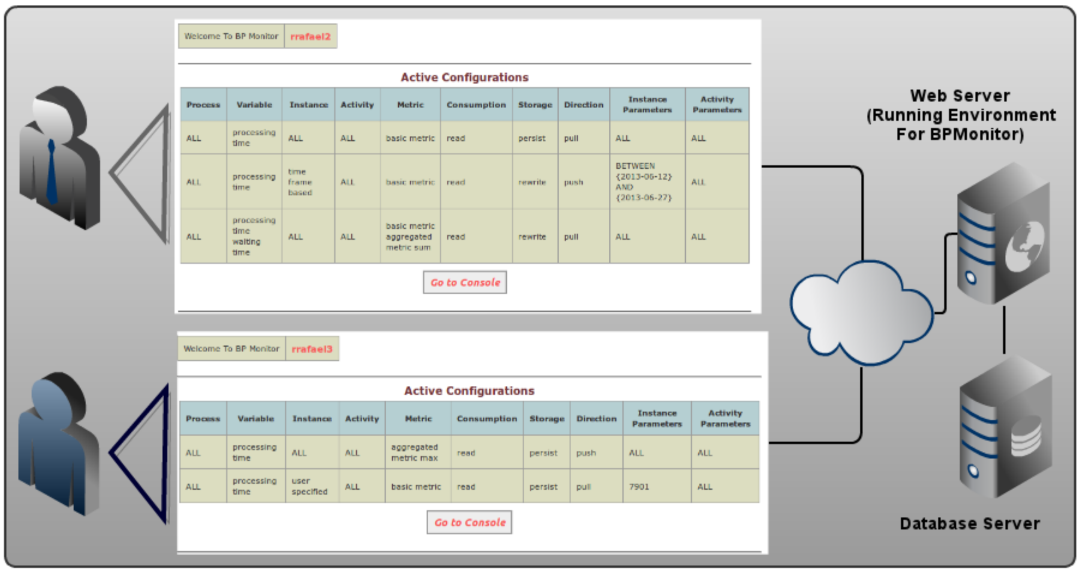
Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters
ALL	processing time	ALL	ALL	basic metric	read	persist	pull	ALL	ALL
ALL	processing time	time frame based	ALL	basic metric	read	rewrite	push	BETWEEN {2013-06-12} AND {2013-06-27}	ALL

Figure 5.6: The multi-dimensional space of the model and BPMonitor Customization Interface.

The previous list confirms that BPMonitor fulfills all the defined functional requirements. On the other hand, as it was previously explained, validation is the process to determine if the system meets the user needs. In this specific case, we are the users of the system, and validation is important for us since it is the key to have a final decision about how useful is our monitoring model.

The reason why we implemented BPMonitor was to prove if the proposed monitoring model could be applied in a real environment and we can validate this by analyzing two main aspects, the former is related to the level of affinity between the theoretical model and the implemented tool. The latter is related to the real benefits that the model is providing to the user through the use of the implemented tool. We are explaining these aspects below:

- The affinity between BPMonitor and the theoretical monitoring model is clear since we use the conceptual architecture to create BPMonitor. However, we need to check the level of affinity between these two in a different level. Figure 5.6 shows now only that BPMonitor



Active Configurations

Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters
ALL	processing time	ALL	ALL	basic metric	read	persist	pull	ALL	ALL
ALL	processing time	time frame based	ALL	basic metric	read	rewrite	push	BETWEEN {2013-06-12} AND {2013-06-27}	ALL
ALL	processing time waiting time	ALL	ALL	basic metric aggregated metric sum	read	rewrite	pull	ALL	ALL

Active Configurations

Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters
ALL	processing time	ALL	ALL	aggregated metric max	read	persist	push	ALL	ALL
ALL	processing time	user specified	ALL	basic metric	read	persist	pull	7901	ALL

Figure 5.7: Customization provided by BPMonitor.

Customization Interface provides an interface that maps directly the multi-dimensional space of our model, but also that this interface is able to manages the different options chosen by the user and groups them in separate and independent configurations, providing a specific monitoring console for each user.

- The monitoring model presented in this thesis is designed to allow users to have a customizable monitoring model that let users to choose different options over both the monitoring variables that were defined as relevant in the context of a BPE and the way monitoring information is retrieved and presented to the user. In figure 5.6 it is possible to see that BPMonitor allows a user to create several configurations with all the possible combinations that our model provides. This is already demonstrating a certain level of customization provided by BPMonitor, however, the monitoring model was intended to support not only customization, but also mass customization and this situation was the main aspect considered in the defined conceptual architecture. In this chapter we have already reviewed how the conceptual architecture was analyzed until the point that we could use it to build BPMonitor, but is BPMonitor really supporting customization in a mass approach?. Figure 5.7 is a clear confirmation that BPMonitor, not only supports several users, but also it manages to support all the different configurations they create, so yes it supports not only customization, but mass customization as well.

In this chapter we have gone through several aspects considered when we are talking about software verification and software validation. Besides checking the theory we also presented the results of a real performance test executed over the monitoring console of the implemented tool. This is really important since we are talking about mass customization, which means that besides checking that the tool allows each user to have a customized monitoring console, it is necessary to validate the "mass" approach by checking that the tool really handles a great load of users. On the basis of all the information presented so far, we will present a conclusion in the next chapter.

Chapter 6

Conclusions

This thesis was intended to analyze the relationship between "Business Process Management", particularly from the management aspect, specifically talking about "Business Process Monitoring", and "Customization". Even when it is possible to find not only a great diversity of research literature, but also a wide range of real life applications for each of them separately, the truth is that their relationship has not been extensively investigated by the literature. Besides, when we extend our vision beyond the theoretical world it is possible to see that current technology provides limited options for business process monitoring customization, and that even when process enactment technology already provides standard consoles, they offer a limited vision of the information. In reality, none of these options represents a customizable model with the required flexibility to meet the needs of all the customers involved in the business processes monitoring context. These arguments are the main reason that our problem definition is to find the answer to the following question: ***Is it possible to define a customizable monitoring model for business processes?***.

Even when we have a clear problem definition, it is true that in order to find the answer that we are looking for, we need to find the answer for more specific questions such as *why do customers need monitoring customization?*, *what do customers expect from a customizable monitoring tool?* and *In case there is a customizable monitoring model, is it possible to adapt it to a real environment?*.

The answer for the first two questions can be found in the chapters where the Literature Study and the Theoretical Model are presented. These chapters take us from the analysis of the literature (which help us to define both the basic relevant metrics in a BPMS and the basic reporting techniques that need to be considered in a reporting tool), and the analysis of the general business process monitoring architecture, to the definition of a set of monitoring variables, which is the basis of the model, and finally to both the formal definition of our monitoring model and its multi-dimensional space of monitoring dimensions and the related options. Even when the complete answers are depicted in these chapters, we can still summarize that the model and its multi-dimensional space are the main part of the answers, simply because they clearly show not only what information customers should expect from a BPMS, but also what are the possible ways, or options, users can get the information. This flexibility that the model provides is the answer to the question why users need the model.

The answer for the last question is presented in the chapters related to the implementation of the tool and its evaluation. In these chapters we clearly demonstrate that the presented monitoring model can go from paper design to a real and tangible implementation. Besides the implementation, the evaluation chapter demonstrate that the tool is able not only to give the flexibility to create different configurations which are presented in a specific monitoring console for each customer, but also to support a great load of users in the system. These two arguments confirm the statement that the model supports not only customization, but mass customization of business process monitoring as well.

Even when the presented analysis and the implemented work is enough to see that indeed it is possible not only to define a customizable monitoring model for business processes, but also to take it to the real world, there is still some future work that we would like to point out. Next section will describe part of the future work we consider should be addressed in order to improve our model.

6.1 Future Work

In this thesis we designed a monitoring model and we took it from theory to reality with accepted results. However, even when our model can be applied in any environment since the conceptual architecture to take it to reality is based on a well supported design pattern, which takes advantage of ”encapsulation” and ”modularization” (these concepts give advantages such as parallel programming, quality, and reusability of modules), the truth is that we are still thinking in the traditional approach of software development that focus on the development of one system at a time. Instead, we should have a broader vision, considering possibilities such as the one described in [5]. According to this paper, ”One increasing trend in software development is the need to develop multiple, similar software products instead of just a single individual product. There are several reasons for this. Products that are being developed for the international market must be adapted for different legal or cultural environments, as well as for different languages, and so must provide adapted user interfaces. Because of cost and time constraints it is not possible for software developers to develop a new product from scratch for each new customer, and so software reuse must be increased”. One of the possible ways to address this trend is to use Software Product Line Engineering, which take us to the concept of ”Software Product Line”. A Software Product Line (SPL) basically consists of a product line architecture and a set of reusable components that can be used in the product line architecture. Some authors regard SPLs as ”software system families”, and this concept is the key of our vision since we are proposing as future work the search of a monitoring tool family instead of a single monitoring tool.

Even when SPL is a different concept, we consider that we are not that far from our vision since we already have a clear theoretical monitoring model for the business process context, and what we need to do now, is to take our model as a reference in order to define a product line architecture and their reusable components.

In the paper [5], it is stated that we can use feature models to define the architecture and components of a SPL. Indeed, it specifies that ”feature models are simple hierarchical models that capture the commonality and variability of Product Lines”. This approach assumes that the components of a system can be taken as their features, given that a feature is ”any characteristic of a system that is relevant for some stakeholder”. Feature models are a powerful tool since they not only have a tree structure where the features are easily represented as the nodes of the tree, but also offer a certain level of flexibility since the arcs of the tree let the user group features in the following categories: ”mandatory”, ”optional”, ”alternative” and ”or”. The popularity to move software development to a new quality of industrial production with the SPL concept has also brought the necessity of new type of tools to support this trend. FeatureIDE is a project, developed under the branch of Eclipse, designed to support the phases of feature-oriented software development for the development of SPLs. Besides the theory, we decided to go further and we created a first draft of the possible feature model of our monitoring model. Figure 6.1 depicts the draft created using FeatureIDE where it is possible to see the monitoring variable ”processing time”, the nodes that indicate that the user have to choose options to select instances, activities, and also the type of metric he or she wants to see either a basic metric or an aggregated value.

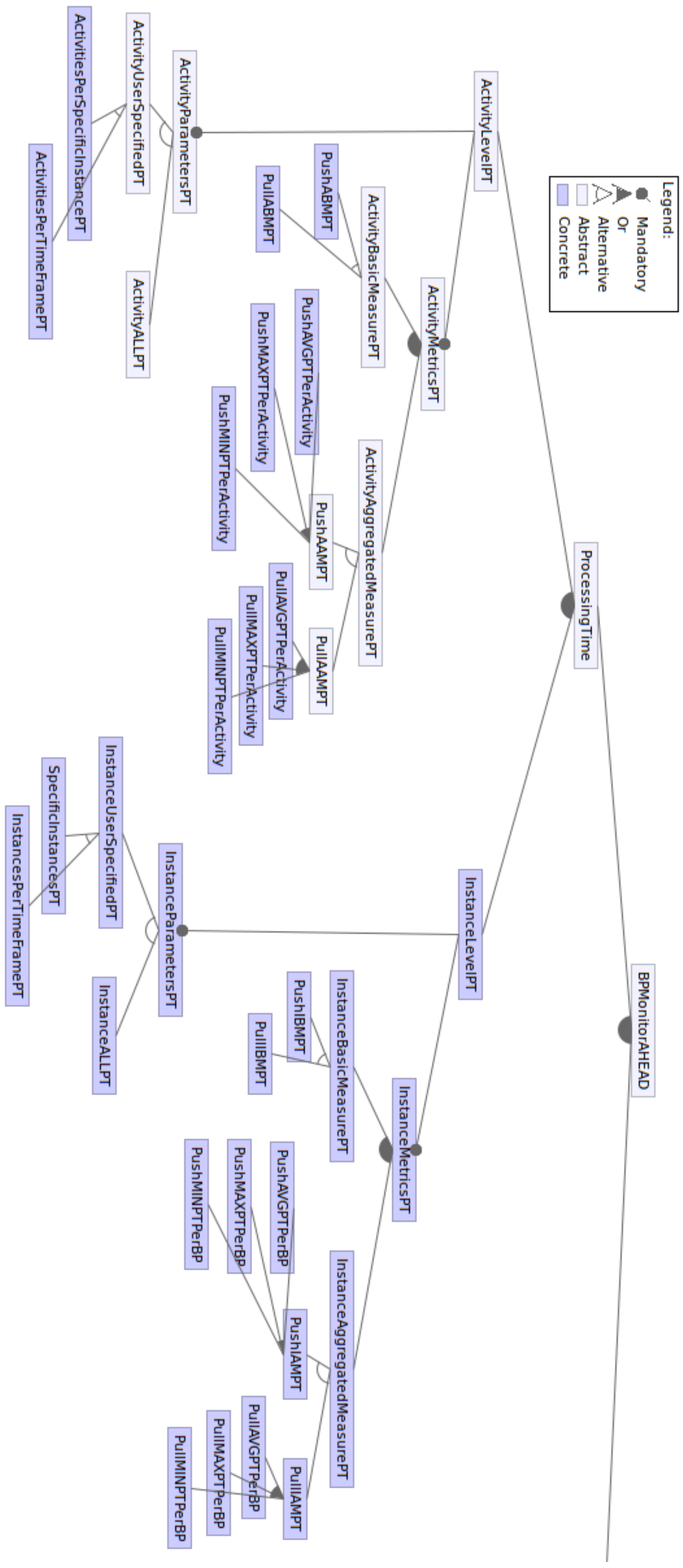


Figure 6.1: Feature Model of BPMonitor using FeatureIDE.

So far we not only presented our vision about the future work for this thesis, but also gave a clear reference of a possible starting point of the proposed future work. It is clear that there is still a lot of work to complete in order to address a mass monitoring system with the approach of software product line, however considering the work completed in this thesis and the literature reviewed about feature models and SPL, we consider that this future work is a promising next step for what we started in this thesis.

Bibliography

[1] Wil M. P. Van Der Aalst. Three good reasons for using a Petri-net-based workflow management system. In *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96),*, pages 179–201, (1996). 26

[2] Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Mathias Weske. Business process management: a survey. In *Proceedings of the 2003 international conference on Business process management*, pages 1–12. Springer-Verlag, 2003. 1, 6

[3] ABPMP. Becoming a BPM Professional. `http://www.abpmp.org/displaycommon.cfm?an=1&subarticlenbr=216`, 2013. [Online; accessed 15-June-2013]. 6

[4] Ian Alexander. Misuse cases help to elicit non-functional requirements. *Computing and Control Engineering Journal*, 14:40–45, 2003. 48

[5] Danilo Beuche and Mark Dalgarno. Software product line engineering with feature models. *ACCU Overload journals*, 2007. 54

[6] Marco Comuzzi, Samuil Angelov, and Jochem Vonk. Patterns to enable mass-customized business process monitoring. In *24th International Conference on Advanced Information Systems Engineering(CAiSE'12)*, 2012. iii, 8, 15, 16

[7] George Cybenko and Brian Brewington. The Foundations of Information Push and Pull. 1998. 13

[8] David Ferraiolo and Richard Kuhn. Role-Based Access Controls. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992. 13

[9] Jan-Philipp Friedenstab, Christian Janiesch, Martin Matzner, and Oliver Muller. Extending BPMN for Business Activity Monitoring. In *Proceedings of the 2012 45th Hawaii International Conference on System Sciences*, pages 4158–4167. IEEE Computer Society, 2012. 9, 10, 13

[10] Gartner. Business Activity Monitoring. In *DEXA Workshops'05*, page URL (June 2013): `http://www.pikos.net/documents/german/Gartner.pdf`, 2003. 7

[11] Nils Glombitza, Dennis Pfisterer, and Stefan Fischer. Integrating wireless sensor networks into web service-based business processes. In *Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*, pages 25–30. ACM, 2009. 32

[12] Daniela Grigori, Fabio Casati, Malu Castellanos, Umeshwar Dayal, Mehmet Sayal, and Ming-Chien Shan. Business process intelligence. *Comput. Ind.*, pages 321–343, 2004. 9, 10, 13

[13] Emily H. Halili. *Apache JMeter: A Practical Beginner's Guide to Automated Testing and Performance Measurement for Your Websites.* Packt Publishing, 2008. 45

[14] Michael Hammer and James Champy. *Reengineering the Corporation: A Manifesto for Business Revolution.* HarperBusiness, 1994. 5

[15] Bjrn Hermans. Desperately Seeking:Helping Hands and Human Touch. *First Monday*, 3(11), 1998. 13

[16] David Hollingsworth. Workflow Management Coalition - The Workflow Reference Model. Technical report, Workflow Management Coalition, 1995. 1

[17] Staffware House. *Staffware Process Suite. Administering the Staffware PE SQL Database, issue 1 edition.* Staffware, 2002. 26

[18] IBM. *IBM Websphere MQ Workflow. Administration Guide.* IBM, 2003. 26

[19] IBM. What is BPM? http://www-01.ibm.com/software/info/bpm/what-is-bpm/, 2013. [Online; accessed 15-June-2013]. 5

[20] William H. Inmon. The Data Warehouse and Data Mining. 39(11):49–50, 1996. 14

[21] John Jeston and Johan Nelis. *Business Process Management Practical Guidelines to Successful Implementations.* Elsevier/Butterworth-Heinemann, 2008. 5, 26

[22] Peter Kung, Claus Hagen, Marisa Rodel, and Sandra Seifert. Business Process Monitoring and Measurement in a Large Bank: Challenges and Selected Approaches. In *DEXA Workshops'05*, pages 955–961, 2005. 7

[23] Soren Lauesen and Houman Younessi. Six styles for usability requirements. In *Proceedings of REFSQ'98*, 1998. 47

[24] Christian Lichka. Strategic Monitoring and Alignment to Achieve Business Process Best Practices. In *DEXA Workshops*, pages 914–918, 2005. 8

[25] Mariska Netjes, Hajo A. Reijers, and Wil M.P. van der Aalst. Supporting the BPM life-cycle with FileNet. In *Proceedings of the Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'06) and the 18th Conference on Advanced Information Systems (CAiSE'06)*, pages 497–508, 2006. 6

[26] Joseph B. Pine. Mass Customization: The New Frontier in Business Competition. 1992. 7

[27] Arun Poduval, Doug Todd, Harish Gaur, Jeremy Bolie, Jerry Thomas, Kevin Geminiuc, Lawrence Pravin, Markus Zirn, Matjaz B. Juric, Michael Cardella, Praveen Ramachandran, Sean Carey, Stany Blanvalet, The Hoa Nguyen, and Yves Coene. *BPEL Cookbook: Best Practices for SOA-based integration and composite applications development.* Packt Publishing, 2006. 31

[28] Ian Roughley. *Starting Struts 2.* Lulu.com, 2007. 29

[29] Gunter Schmidt and Wilbert E. Wilhelm. Strategic, Tactical and Operational Decisions in Multi-national Logistics Networks: A Review and Discussion of Modeling Issues. *International Journal of Production Research*, 38:38–7, 2000. 8

[30] Giovani Da Silveira, Denis Borenstein, and Flávio S. Fogliatto. Mass customization: Literature review and research directions. *International Journal of Production Economics*, 72:1–13, 2001. 1

[31] Bruce Silver. BPMS Report: Understanding and Evaluating BPM Suite. *BPMInstitute.org*, pages URL (June 2013): http://www.bpminstitute.org/resources/articles/bpms--watch--understanding--and--evaluating--bpm--suites, 2006. 5

[32] Appian Software. What is BPM. http://appiancorp.com/Literature/pdfs/whatisbpm.pdf, 2006. [Online; accessed 15-June-2013]. 6

[33] Brian Squire, Jeff Readman, Steve Brown, and John Bessant. Mass customization: the key to customer value? *Production Planning and Control*, pages 459–471, 2004. 7

[34] Michael zur Muehlen and Robert Shapiro. Business Process Analytics. In *Handbook on Business Process Management 2*, pages 137–157. Springer, 2010. 11, 12, 13

Appendix A

Information about BPMonitor

A.1 BPMonitor Web Interfaces

This appendix will present the different web interfaces that will be displayed to the user while using BPMonitor.

The following steps represent the case when the user use BPMonitor to create a new configuration and then to open the monitoring console to see the results:

1. User "rrafael" enters his credentials into the system. This will be done using the interface of figure A.1.
2. User accesses the Customization Interface which is depicted in figure A.2.
3. User creates a new configuration using the Customization Interface. This is depicted in figure A.3.
4. User updates the new configuration into the active configurations of the Customization Interface. This is depicted in figure A.4.
5. User jumps into the Monitoring Console which shows the active configurations and the list of links to specific result interfaces. This is depicted in figure A.5.
6. User opens the first specific result which gives information in an activity level of detail. This is depicted in figure A.6.
7. User opens the second specific result which gives information in an instance level of detail. This is depicted in figure A.7.

We need to clarify also how BPMonitor offers the option of information pull and information push. Figure A.8 depicts how BPMonitor handles the two options to retrieve the data.

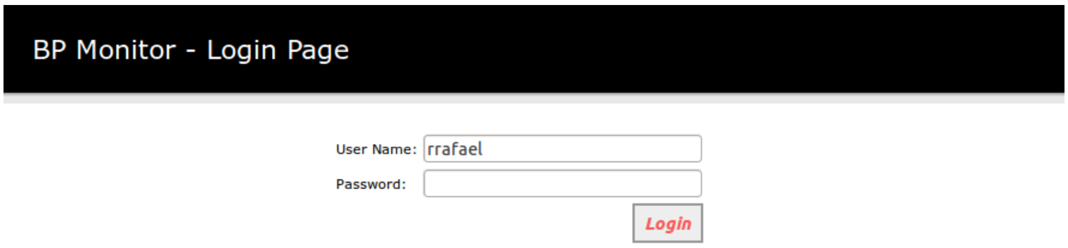
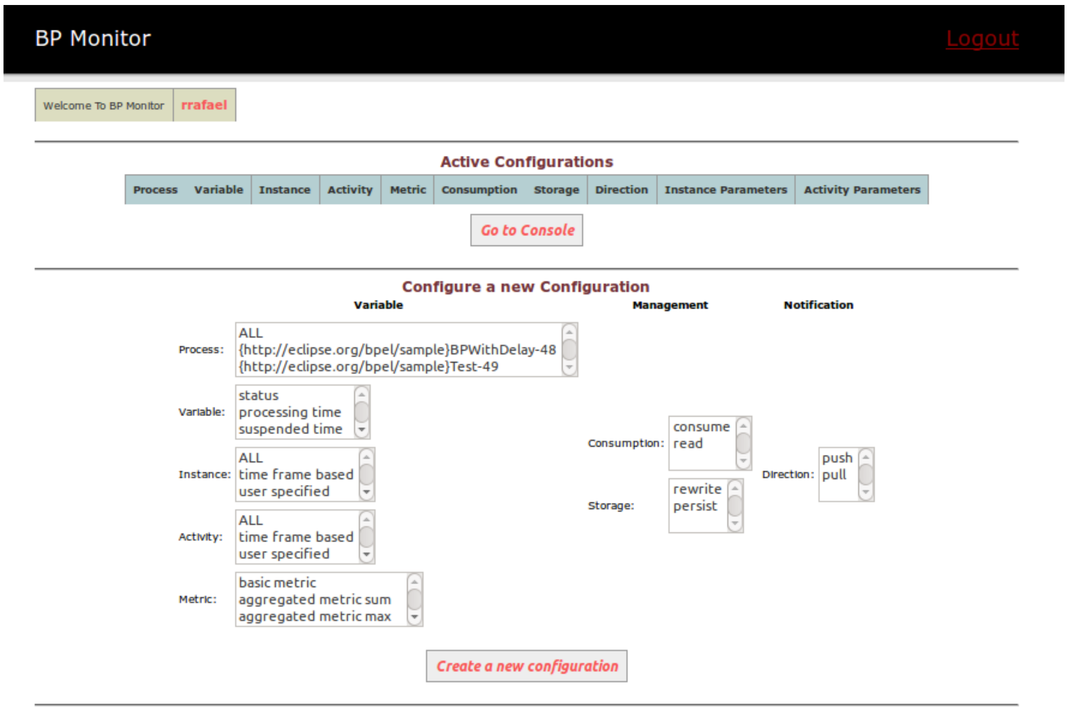


Figure A.1: Login Interface



Process: ALL / {http://eclipse.org/bpel/sample}BPWithDelay-48 / {http://eclipse.org/bpel/sample}Test-49

Variable: status / processing time / suspended time

Instance: ALL / time frame based / user specified

Activity: ALL / time frame based / user specified

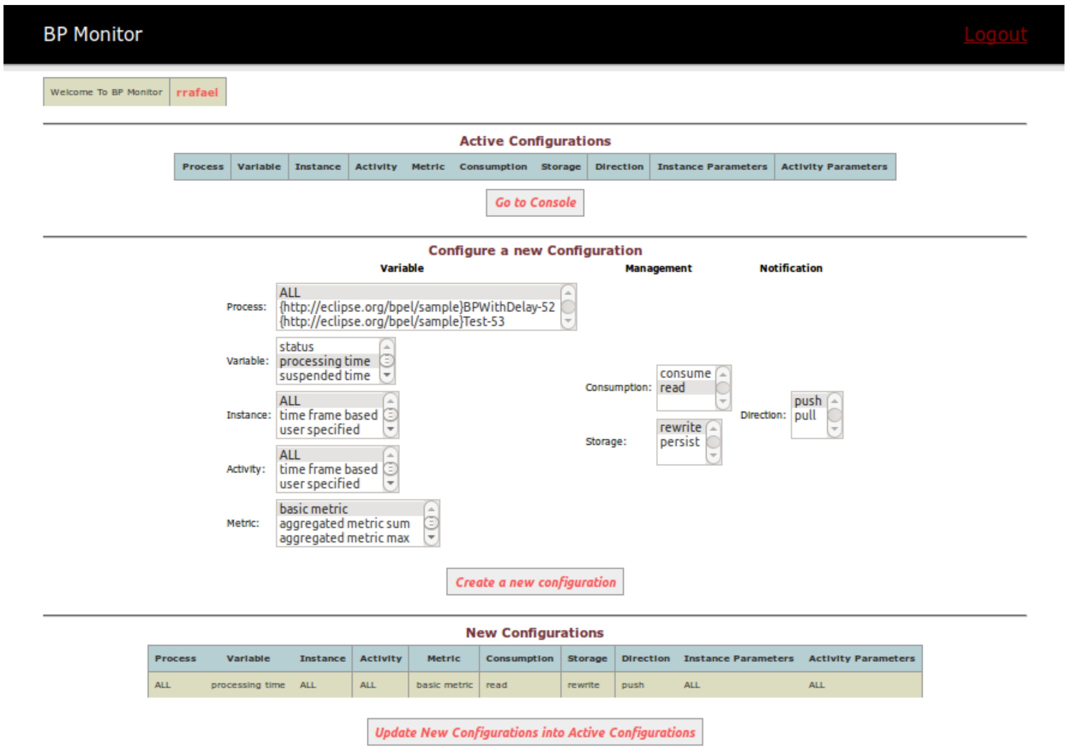
Metric: basic metric / aggregated metric sum / aggregated metric max

Consumption: consume / read

Storage: rewrite / persist

Direction: push / pull

Figure A.2: Customization Interface



Active Configurations

Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters

Configure a new Configuration

Variable — Management — Notification

Process: ALL / {http://eclipse.org/bpel/sample}BPWithDelay-52 / {http://eclipse.org/bpel/sample}Test-53

Variable: status / processing time / suspended time

Instance: ALL / time frame based / user specified

Activity: ALL / time frame based / user specified

Metric: basic metric / aggregated metric sum / aggregated metric max

Consumption: consume / read

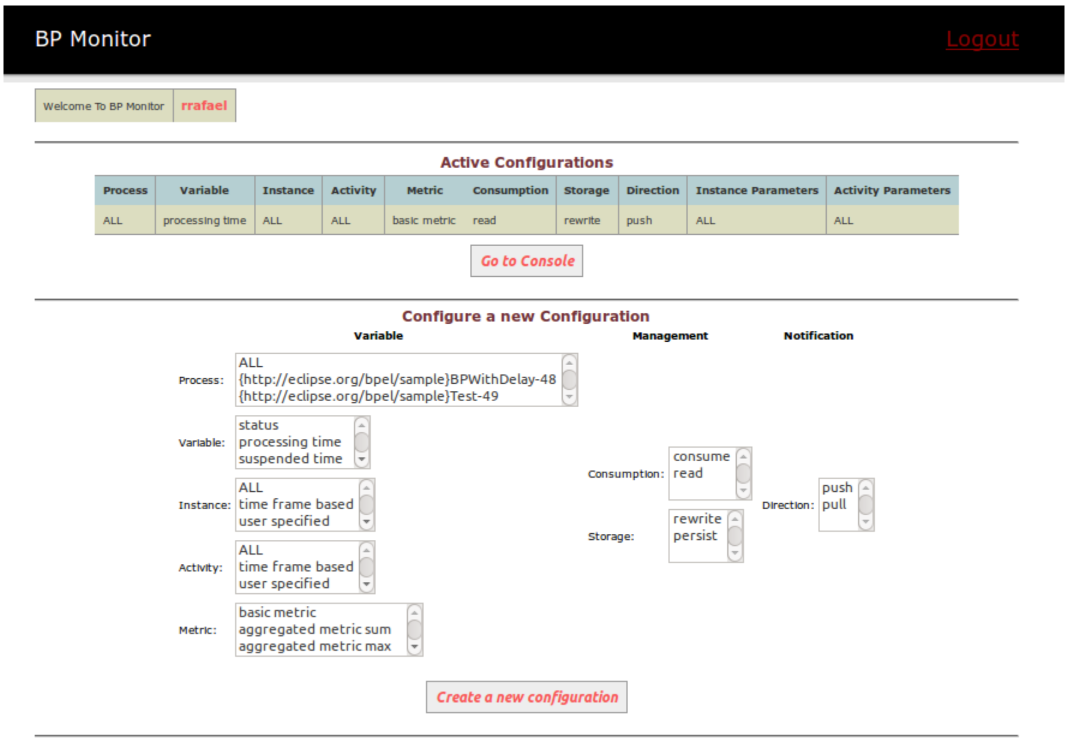
Storage: rewrite / persist

Direction: push / pull

New Configurations

Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters
ALL	processing time	ALL	ALL	basic metric	read	rewrite	push	ALL	ALL

Figure A.3: Customization Interface - creating new configuration



Active Configurations

Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters
ALL	processing time	ALL	ALL	basic metric	read	rewrite	push	ALL	ALL

Configure a new Configuration

Variable — Management — Notification

Process: ALL / {http://eclipse.org/bpel/sample}BPWithDelay-48 / {http://eclipse.org/bpel/sample}Test-49

Variable: status / processing time / suspended time

Instance: ALL / time frame based / user specified

Activity: ALL / time frame based / user specified

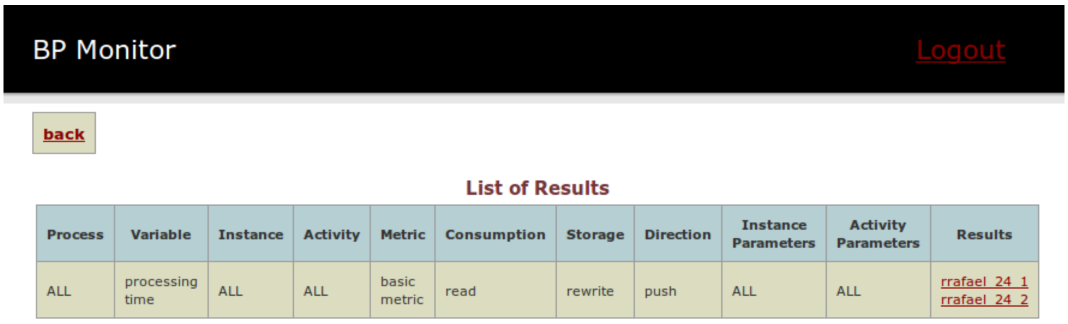
Metric: basic metric / aggregated metric sum / aggregated metric max

Consumption: consume / read

Storage: rewrite / persist

Direction: push / pull

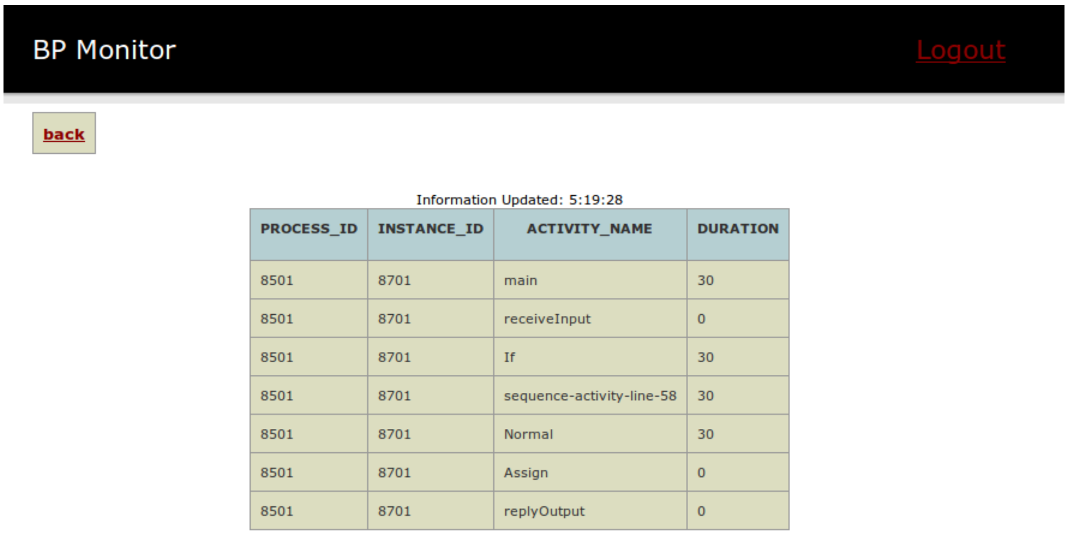
Figure A.4: Customization Interface - updating new configuration into active configurations



List of Results

Process	Variable	Instance	Activity	Metric	Consumption	Storage	Direction	Instance Parameters	Activity Parameters	Results
ALL	processing time	ALL	ALL	basic metric	read	rewrite	push	ALL	ALL	rrafael_24_1 rrafael_24_2

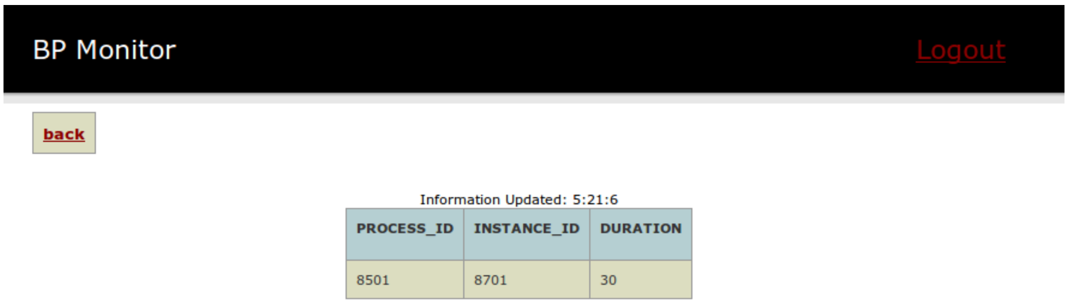
Figure A.5: Monotoring Console - list of results



Information Updated: 5:19:28

PROCESS_ID	INSTANCE_ID	ACTIVITY_NAME	DURATION
8501	8701	main	30
8501	8701	receiveInput	0
8501	8701	If	30
8501	8701	sequence-activity-line-58	30
8501	8701	Normal	30
8501	8701	Assign	0
8501	8701	replyOutput	0

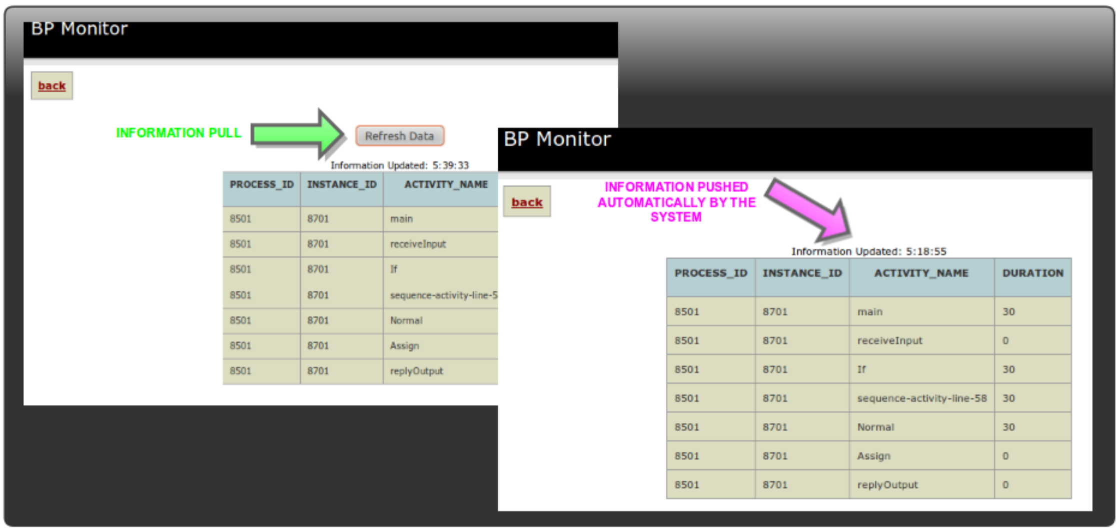
Figure A.6: Monotoring Console - specific result (activity level of detail)



Information Updated: 5:21:6

PROCESS_ID	INSTANCE_ID	DURATION
8501	8701	30

Figure A.7: Monotoring Console - specific result (instance level of detail)



Information Updated: 5:39:33

PROCESS_ID	INSTANCE_ID	ACTIVITY_NAME
8501	8701	main
8501	8701	receiveInput
8501	8701	If
8501	8701	sequence-activity-line-5
8501	8701	Normal
8501	8701	Assign
8501	8701	replyOutput

Information Updated: 5:18:55

PROCESS_ID	INSTANCE_ID	ACTIVITY_NAME	DURATION
8501	8701	main	30
8501	8701	receiveInput	0
8501	8701	If	30
8501	8701	sequence-activity-line-58	30
8501	8701	Normal	30
8501	8701	Assign	0
8501	8701	replyOutput	0

Figure A.8: Monotoring Console - Information Pull vs Information Push

A.2 BPMonitor database script

In this section we present the sql script that we use to create the internal database that supports BPMonitor. This script is intended to be executed over a MySQL server environment.

```
CREATE DATABASE BPMONITOR;

USE BPMONITOR;

CREATE TABLE BPMON_USER
(ID BIGINT NOT NULL AUTO_INCREMENT,
USERNAME VARCHAR(255),
PASSWORD VARCHAR(255),
PRIMARY KEY (ID));

CREATE TABLE BPMON_CONFIGURATIONS
(ID BIGINT NOT NULL AUTO_INCREMENT,
USER_ID BIGINT,
PROCESSOP VARCHAR(255),
VARIABLEOP VARCHAR(255),
INSTANCEOP VARCHAR(255),
ACTIVITYOP VARCHAR(255),
METRICOP VARCHAR(255),
CONSUMPTIONOP VARCHAR(255),
STORAGEOP VARCHAR(255),
DIRECTIONOP VARCHAR(255),
AVAILABLE BIGINT,
TSTAMP DATETIME,
INSTANCEPARAM VARCHAR(255),
ACTIVITYPARAM VARCHAR(255),
PRIMARY KEY (ID)
);

CREATE TABLE BPMON_VARIABLE_QUERY
(ID BIGINT NOT NULL AUTO_INCREMENT,
VARIABLE_NAME VARCHAR(255),
MEASURE_TYPE VARCHAR(255),
QUERYHEADERSTR VARCHAR(255),
QUERYSTR VARCHAR(2000),
PRIMARY KEY (ID)
);

insert into BPMON_VARIABLE_QUERY(VARIABLE_NAME, MEASURE_TYPE,
QUERYHEADERSTR, QUERYSTR)
values (
'processing time',
'basic metric',
'PROCESS_ID,INSTANCE_ID,ACTIVITY_NAME,DURATION',
'select a.PROCESS_ID, a.INSTANCE_ID as INSTANCE_ID,
a.activityName as ACTIVITY_NAME, TIMESTAMPDIFF(SECOND, startTime,
endTime) as DURATION from (SELECT trim(replace(replace
(SUBSTRING(DETAIL,INSTR(DETAIL, #ActivityName#) + 14,
(INSTR(DETAIL, #ActivityType#) - (INSTR(DETAIL, #ActivityName#)
+ 14))),#diagN#,##),#diagT#, ##)) as activityName, TSTAMP startTime,
EVENT_ID, INSTANCE_ID, PROCESS_ID FROM ODE_EVENT WHERE INSTANCE_ID
IN (@INSTANCES@) AND TYPE LIKE #ActivityExecStartEvent# AND
INSTR(DETAIL, #ActivityName#) != 0 ORDER BY INSTANCE_ID, TSTAMP,
EVENT_ID) a, (SELECT trim(replace(replace(SUBSTRING(DETAIL,
INSTR(DETAIL, #ActivityName#) + 14, (INSTR(DETAIL, #ActivityType#) -
(INSTR(DETAIL, #ActivityName#) + 14))),#diagN#,##),#diagT#, ##))
as activityName,TSTAMP endTime, EVENT_ID, INSTANCE_ID, PROCESS_ID
FROM ODE_EVENT WHERE INSTANCE_ID IN (@INSTANCES@) AND TYPE LIKE
#ActivityExecEndEvent# AND INSTR(DETAIL, #ActivityName#)  != 0
ORDER BY INSTANCE_ID, TSTAMP, EVENT_ID) b WHERE a.activityName =
b.activityName AND a.INSTANCE_ID = b.INSTANCE_ID AND
a.PROCESS_ID = b.PROCESS_ID ORDER BY a.INSTANCE_ID, a.EVENT_ID;'
);

insert into BPMON_VARIABLE_QUERY(VARIABLE_NAME, MEASURE_TYPE,
QUERYHEADERSTR, QUERYSTR)
values (
'processing time',
'basic metric',
'PROCESS_ID,INSTANCE_ID,DURATION',
'select a.PROCESS_ID, a.INSTANCE_ID,
TIMESTAMPDIFF(SECOND, startTime, endTime) as DURATION
from (select PROCESS_ID, INSTANCE_ID, TSTAMP AS startTime
FROM ODE_EVENT WHERE TYPE = #NewProcessInstanceEvent# and
INSTANCE_ID IN (@INSTANCES@)) a,(select PROCESS_ID, INSTANCE_ID,
TSTAMP AS endTime FROM ODE_EVENT WHERE TYPE = #ProcessCompletionEvent#
and INSTANCE_ID IN (@INSTANCES@)) b where a.PROCESS_ID = b.PROCESS_ID
AND a.INSTANCE_ID = b.INSTANCE_ID;'
);

insert into BPMON_VARIABLE_QUERY(VARIABLE_NAME, MEASURE_TYPE,
QUERYHEADERSTR, QUERYSTR)
values (
'processing time',
'aggregated metric',
'PROCESS_ID, ACTIVITY_NAME,',
'select PROCESS_ID, ACTIVITY_NAME, @HEADER_RESULTS@ FROM
(select a.PROCESS_ID, a.INSTANCE_ID as INSTANCE_ID, a.activityName
as ACTIVITY_NAME, TIMESTAMPDIFF(SECOND, startTime, endTime)
as DURATION from (SELECT trim(replace(replace(SUBSTRING(DETAIL,
INSTR(DETAIL, #ActivityName#) + 14, (INSTR(DETAIL, #ActivityType#)
- (INSTR(DETAIL, #ActivityName#) + 14))),#diagN#,##),#diagT#, ##))
as activityName,TSTAMP startTime, EVENT_ID, INSTANCE_ID,
PROCESS_ID FROM ODE_EVENT WHERE INSTANCE_ID IN (@INSTANCES@) AND
TYPE LIKE #ActivityExecStartEvent# AND INSTR(DETAIL, #ActivityName#)
!= 0 ORDER BY INSTANCE_ID, TSTAMP, EVENT_ID) a, (SELECT
trim(replace(replace(SUBSTRING(DETAIL, INSTR(DETAIL, #ActivityName#)
+ 14, (INSTR(DETAIL, #ActivityType#) - (INSTR(DETAIL, #ActivityName#)
+ 14))),#diagN#,##),#diagT#, ##)) as activityName,TSTAMP endTime,
EVENT_ID, INSTANCE_ID, PROCESS_ID FROM ODE_EVENT WHERE
INSTANCE_ID IN (@INSTANCES@) AND TYPE LIKE #ActivityExecEndEvent# AND
INSTR(DETAIL, #ActivityName#) != 0 ORDER BY INSTANCE_ID, TSTAMP,
EVENT_ID) b WHERE a.activityName = b.activityName AND
a.INSTANCE_ID = b.INSTANCE_ID AND a.PROCESS_ID = b.PROCESS_ID
```

```
ORDER BY a.INSTANCE_ID, a.EVENT_ID) c
GROUP BY PROCESS_ID, ACTIVITY_NAME;'
);

insert into BPMON_VARIABLE_QUERY(VARIABLE_NAME, MEASURE_TYPE,
QUERYHEADERSTR, QUERYSTR)
values (
'processing time',
'aggregated metric',
'PROCESS_ID,',
'select PROCESS_ID, @HEADER_RESULTS@ FROM (select a.PROCESS_ID,
a.INSTANCE_ID, TIMESTAMPDIFF(SECOND, startTime, endTime) as DURATION
from (select PROCESS_ID, INSTANCE_ID, TSTAMP AS startTime
FROM ODE_EVENT WHERE TYPE = #NewProcessInstanceEvent# and
INSTANCE_ID IN (@INSTANCES@)) a, (select PROCESS_ID, INSTANCE_ID,
TSTAMP AS endTime FROM ODE_EVENT WHERE TYPE = #ProcessCompletionEvent#
and INSTANCE_ID IN (@INSTANCES@)) b where a.PROCESS_ID = b.PROCESS_ID
AND a.INSTANCE_ID = b.INSTANCE_ID) c GROUP BY PROCESS_ID;'
);
```