Analysis and implementation of patterns for mass customization in business process monitoring

Yu, C.

Award date:
2012

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
Analysis and implementation of patterns for mass customization in business process monitoring

Chen Yu

Business Information Systems

Supervisor:

dr. M. (Marco) Comuzzi

Examination committee:

dr. M. (Marco) Comuzzi

dr.ir. ITP (Irene) Vanderfeesten

dr. FM (Fabrizio) Maggi

Eindhoven, August 2012
# Table of Contents

1  Introduction .......................................................................................................................... 3  
   1.1  The focus of this thesis ................................................................................................. 3  
   1.2  The structure of this thesis ......................................................................................... 3  
2  Literature review and customization infrastructure ............................................................. 4  
   2.1  Business process configuration vs. customization ........................................................ 5  
   2.2  Aspects that can be customized ................................................................................... 8  
      2.2.1  Behavior customization ......................................................................................... 8  
      2.2.2  Quality of Service (QoS) customization ............................................................... 13  
      2.2.3  Resource customization ....................................................................................... 16  
   2.3  Mass-customizer capabilities ....................................................................................... 19  
   2.4  Proposed business process monitoring customization infrastructure ....................... 20  
3  Monitoring dimensions and patterns .................................................................................... 22  
   3.1  Monitoring dimensions ............................................................................................... 23  
   3.2  Monitoring Patterns ..................................................................................................... 26  
   3.3  Combining patterns to design Monitoring infrastructure ........................................... 31  
4  Implementation and Architecture .......................................................................................... 35  
   4.1  Implementation ............................................................................................................. 35  
      4.1.1  Class Diagram ....................................................................................................... 35  
      4.1.2  Running the program ............................................................................................ 38  
      4.1.3  Pattern explanation ............................................................................................... 40  
      4.1.4  Analysis of the implementation ............................................................................ 41  
   4.2  Architecture .................................................................................................................. 43  
      4.2.1  Business Process Execution Language for Web Services (BPEL) ....................... 44  
      4.2.2  Yet Another Workflow Language (YAWL) .......................................................... 45  
      4.2.3  Other business process engines ............................................................................ 48  
      4.2.4  On-line advertising example ............................................................................... 49  
      4.2.5  Analysis of the proposed architecture .................................................................. 50  
4.3  Quality and Evaluation .................................................................................................... 52  
   4.3.1  Criteria ..................................................................................................................... 52  
   4.3.2  Analysis ................................................................................................................... 53
1 Introduction

1.1 The focus of this thesis
This thesis is about the mass customization with the application in the business process monitorability domain, i.e., about the design and implementation of monitorability customization infrastructure. More concisely speaking, the thesis focuses on designing patterns to support mass customization of business process monitoring.

1.2 The structure of this thesis
The structure of this document is as follows, which is shown in Figure 1. In Chapter 2, a literature study about mass customization with application in business process field is conducted. Two points are concluded, one is the substantial lack of literature on mass customization in business process management; the other is there is a need to define the monitoring patterns. In the end, the definition and implementation of patterns to support mass customization in business process monitoring are proposed. Chapter 3 describes monitoring dimensions, in each of which some patterns are defined in Petri Net. Based on the dimensions and patterns, in chapter 4, the architecture of the business process monitorability customization infrastructure and the implementation of the framework are presented. Then we offer a framework of criteria which is used to evaluate the implementation. Moreover, the pros and cons are analyzed. Chapter 5 contains the conclusions of this thesis, in which we emphasis the concept of business process monitorability customization.

![Figure 1 - Roadmap of the thesis](image-url)
2 Literature review and customization infrastructure

At the beginning, we will conduct a literature research about both mass customization and business process monitorability.

Mass customization (MC) is defined as “developing, producing, marketing and delivering affordable goods and services with enough variety and customization that nearly everyone finds exactly what they want” [1]. It can be comprehended either broadly or narrowly. In a broad vision, firstly introduced by Davis [2], MC is defined as the ability to provide individually designed products and services to every customer through high process agility, flexibility and integration [3–5]. Many other authors propose similar but narrower, more practical concepts. They promote MC as an information system that uses flexible processes and organizational structures to deliver variety of products and services that meet individual requirements from different customers, at a cost near that of mass produced items [5–9]. It has been widely and successfully used by many companies, especially in manufacturing and service industries. The aim of mass customization is to achieve a tremendous increase in variety and customization without a corresponding increase in costs, or in one sentence, to challenge the one-size-fits-all assumption of mass production.

Mass-customization is one of the key issues surrounding Enterprise software since 1980s, such as ERP. [10] According to the research 1, large companies with over $500 million annual revenues are more likely to customize their software. One of the primary drivers of enterprise software customization is lack of direction regarding business requirements. Providing clear-defined requirements, a project team can always implement the software to meet every requirement from the company. Meanwhile, establishing strong project controls and governance is crucial for the project; otherwise, the project team will be more likely to customize every requirement without prioritizing, rationalizing or identifying potential solution, which is considered to be very costly and inefficient. Enterprise software customization concerns more about the IS implementation phase, most of business requirements and objectives are met via configuration and set-up (such as parameters, fields and workflows) together with customization, which requires changes to the source code and also requires a higher level of technical support. Compared to enterprise software customization, mass-customization provides a so-called “on-the-fly” method, i.e. right before being used by the customer. Therefore, enterprise software customization is out of scope for this thesis.

2.1 Business process configuration vs. customization

Business process customization is usually opposed to business process configuration. Process configuration deals with the problem of managing families of business processes, i.e. business processes that are similar to one another in many ways, yet differ in some other ways from one organization, project or industry to another. This problem arises for example in the context of multinational companies that need to localize their business processes to different legislations, compliance regulations, quality requirements, etc. It also manifests itself in the context of acquisition projects, where an organization needs to merge their own processes with the ones of the acquired organization. As defined in [11–13], the aim of business process configuration is to design reference process models capturing behavior of a set of process variants serving the same business goal.

In order to exploit configurable process models in the Process-Aware Information Systems (PAIS) lifecycle, the traditional design phase is split into two phases: one where the configurable process model is designed from a collection of selected process variants, and another where the model is actually configured and individualized to fit a particular setting, as shown in the right part of Figure 4.
Consider a service providing company that provides complaint handling system to different companies. The process starts when the new handling request arrives. After an initial check is made, the request can be divided into two categories, one is from inside the company and another is coming from outside the company. If the request is inside the company, a simple check will be executed and a notification letter will be sent. When dealing the complaints outside the company, a complex check will be executed and a notification letter will be sent.

The service providing company gives the corporate the option to configure business process variant. The generic business process model is shown in Figure 5 (a). For example, a company A, who only wants to deal with the complaints inside the company, will configure and individualize right after the design phase, which can be seen in Figure 5 (b) and (c). Then an individual business process model is derived. In the implementation phase, the running business process instances are all based on the derived business process model built in last step, which is shown in Figure 5 (d).

However, if we still consider the company A, after sometime the company finds due to the simplification of the complaints process the customer satisfaction level decreases. As a result, the company wants to improve the customer satisfaction level by improving the business process with more handling steps. Since the basic business process model didn’t provide appropriate business process variants to capture this special requirement,
configuration technique did not well solve the problem. To set down this problem, the business process injection may help, which is known as the business process customization, which is shown in Figure 6.

From the example, we can conclude that if there are too many possible process variants, business process configuration approach may lead to very complicated models, which are quite costly and hard to be understood by process customers. However, business process customization gives the customer the options to customize their own business process without the limits of business process variants. As a result, we can argue that business process customization approach better suit mass-customization compared to business process configuration approach. In a nutshell, configuration remains a design-time concern.
Aspects that can be customized

Given a standard version of base business process, customization can be ranged from business process variants customization [14, 15, 16] e.g. the option for the consumer to customize the business process variants, such as skip, redo, or cancel activities, to QoS customization [17, 18] e.g. the option for the consumer to pick up a certain level of guaranteed security in financial transactions, or resource (such as objects and roles in the business process relations) customization [13, 19] e.g. the opportunity for clients to directly choose which resources will be used in the process, or monitorability customization, e.g. the option for clients to specify their own monitoring requirements at runtime. In the following few sub-sections, the aspects that can be customized will be illustrated in a decreasing investigation-level order.

2.2.1 Behavior customization

Business process variants can be treated as a core issue in the evolving information systems. No matter how good an information system is, without the ability to capture changing environment, it is bound to fail or become useless. By providing customers the option to customize the business process variants, it will be beneficial to both service providers and consumers.

In paper [14], by defining a set of interference options, called I- and PI-options, which are shown in Table 1, Table 2 and Table 3, it can be made available to a consumer organization to exert control over the base business process. In paper [15], they propose an extended Provop framework (PROcess Variants by OPtions) for modeling and managing large collections of process variants in an adequate way. These two frameworks solve the problem that business processes cannot capture changes in modern business relations. In paper [16], they propose a framework employing OVM (Orthogonal Variability Model) language to solve the problem of customization and deployment support for SaaS applications.

2.2.1.1 I- and PI-options framework

S. Angelov, J. Vonk, P. Grefen, and K. Vidyasankar [14] present a framework for the support of cross-organizational process control in process-intensive business collaborations. The framework consists of two key elements, which are fundamental for the design of solutions for cross-organizational process control. I-options and PI-options are proposed for the control primitives on activities and processes, respectively. The control primitives that are offered to the service consumer at an IP are called “Interference options” (I-options). The control primitives that are offered to the complete process (or a sub-process) are called “Process I-options” (PI-options).
I-options can be classified into three groups as shown in Table 1, i.e. I-options that are applicable before the execution of an activity (group 1), I-options that are applicable during the execution of an activity (group 2), and I-options that are applicable after the execution of an activity (group 3). Each I-option is parameterized upon invocation. A parameter common for all I-options is the activity to which the I-option is applied. Other parameters may be provided, e.g. duration (for DELAY and PAUSE). Based on the basic I-options defined in Table 1, complex I-options (combinations of several basic I-options) can be defined, part of which is shown in Table 2.

<table>
<thead>
<tr>
<th>I-option</th>
<th>Constituent I-options</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSTPONE</td>
<td>DELAY+PROCEED</td>
<td>The execution of an activity is postponed.</td>
</tr>
<tr>
<td>RESTART</td>
<td>PART-RESET+START</td>
<td>A started activity is stopped and is started from the beginning.</td>
</tr>
<tr>
<td>PART-REDO</td>
<td>PART-UNDO+START</td>
<td>A started activity is stopped, undone, and started again.</td>
</tr>
<tr>
<td>TERMINATE</td>
<td>PART-UNDO+SKIP</td>
<td>A started activity is stopped and undone. The control flow is passed to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the next activity.</td>
</tr>
<tr>
<td>RETRY</td>
<td>RESET+START</td>
<td>An ended activity is started from the beginning.</td>
</tr>
<tr>
<td>REDO</td>
<td>UNDO+START</td>
<td>An ended activity is undone and started again</td>
</tr>
</tbody>
</table>

Table 1 - List of I-options

<table>
<thead>
<tr>
<th>I-option</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>The execution of an activity is started</td>
</tr>
<tr>
<td>DELAY/PROCEED</td>
<td>Starting execution of an activity is delayed / continued.</td>
</tr>
<tr>
<td>SKIP</td>
<td>The execution of a non-started activity is skipped.</td>
</tr>
<tr>
<td>PAUSE/CONTINUE</td>
<td>The execution of a started activity is paused / resumed.</td>
</tr>
<tr>
<td>CANCEL</td>
<td>The execution of a started activity is stopped and the activity is</td>
</tr>
<tr>
<td></td>
<td>considered ended. Partial results from the execution of the activity</td>
</tr>
<tr>
<td></td>
<td>remain.</td>
</tr>
<tr>
<td>PART-RESET</td>
<td>The execution of a started activity is stopped and the activity is</td>
</tr>
<tr>
<td></td>
<td>put back in its ready state without undoing any of the work that has</td>
</tr>
<tr>
<td></td>
<td>been performed.</td>
</tr>
<tr>
<td>PART-UNDO</td>
<td>The execution of a started activity is stopped, what has been done</td>
</tr>
<tr>
<td></td>
<td>is undone, and the activity is put back in its ready state.</td>
</tr>
<tr>
<td>RESET</td>
<td>An activity that has ended is put back in its ready state. Results</td>
</tr>
<tr>
<td></td>
<td>from previous execution are not undone.</td>
</tr>
<tr>
<td>UNDO</td>
<td>An activity that has ended is put back in its ready state, after the</td>
</tr>
<tr>
<td></td>
<td>results from the previous execution are undone.</td>
</tr>
</tbody>
</table>

Table 2 - Sample list of complex I-options
An external process can be seen as an aggregation of the activities at the external level to a single, higher-level activity. Consequently, a PI-option can be viewed as an I-option defined on an activity (the process). That is why the set of PI-options includes the set of I-options that defined in Table 1. Driven by our goal to allow the definition of process controls that influence the optional execution and the order of execution of activities, the extension allows the definition of sub-processes of external activities without providing details on the control flow between these activities, i.e. a sub-process can contain a non-complete process specification from which the control flow is omitted. So, in addition to the activity I-options, two new control primitives that operate over sets of activities can be defined, i.e. \textsc{choice} and \textsc{order} that can be used to control which and/or in what order the activities in a sub-process need to be executed. In Table 3, PI-options are listed and the semantics at a process level are explained.

<table>
<thead>
<tr>
<th>PI-option</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td></td>
</tr>
<tr>
<td>START</td>
<td>The execution of a process is started</td>
</tr>
<tr>
<td>DELAY/PROCEED</td>
<td>Starting execution of a process is delayed / continued.</td>
</tr>
<tr>
<td>SKIP</td>
<td>The execution of a non-started process is skipped</td>
</tr>
<tr>
<td>CHOICE</td>
<td>From the set of activities in the process one or more are chosen to be executed.</td>
</tr>
<tr>
<td>ORDER</td>
<td>The order of execution of (some of) the activities in the process is set.</td>
</tr>
<tr>
<td>Group 2</td>
<td></td>
</tr>
<tr>
<td>PAUSE/CONTINUE</td>
<td>The execution of a started process is paused / resumed.</td>
</tr>
<tr>
<td>CANCEL</td>
<td>The execution of a started process is terminated. Partial results from the execution of the process remain</td>
</tr>
<tr>
<td>PART-RESET</td>
<td>The execution of a started process is stopped and the process is put back in its ready state without undoing any of the work that has been performed.</td>
</tr>
<tr>
<td>PART-UNDO</td>
<td>The execution of a started process is stopped, what has been done is undone, and the activity is put back in its ready state.</td>
</tr>
<tr>
<td>Group 3</td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>A process that has ended is put back in its ready state. Results from previous execution are not undone.</td>
</tr>
<tr>
<td>UNDO</td>
<td>A process that has ended is put back in its ready state, after the results from the previous execution are undone.</td>
</tr>
</tbody>
</table>

Table 3 - List of PI-options

2.2.1.2 Provop framework

In Provop approach, A. Hallerbach, T. Bauer, and M. Reichert [15] propose the so called Provop framework in different context to a large spectrum of processes from different application domains. Provop supports various kinds of patterns: \textsc{insert} fragment, \textsc{delete} fragment, \textsc{modify} fragment and \textsc{move} fragment, which are detailed explained in Table 4.

1. INSERT-Operation
Symbol: ▲
Purpose: Addition of *process fragments* (A process fragment consists of at least one process element, e.g. activity nodes or control edges)
Parameters:
- Process Fragment to be added with entries and exits marked by adjustment points
- Target position of the process fragment within the base process, marked by adjustment points for entries and exits
- Mapping between entries and exits of the added fragment to the target position within base processes (i.e., mapping of the respective adjustment points)

<table>
<thead>
<tr>
<th>2. DELETE-Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol: ▼</td>
</tr>
<tr>
<td>Purpose: Removal of process elements</td>
</tr>
<tr>
<td>Parameters:</td>
</tr>
<tr>
<td>- Process fragment to be deleted with entries and exits marked by adjustment points</td>
</tr>
<tr>
<td>- Alternatively: deleting single elements by referring to their ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3. MOVE-Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol: ▮</td>
</tr>
<tr>
<td>Purpose: Change execution order of activities</td>
</tr>
<tr>
<td>Parameters:</td>
</tr>
<tr>
<td>- Process fragment to be moved with entries and exits marked by adjustment points</td>
</tr>
<tr>
<td>- Target position of the process fragment marked by adjustment points</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. MODIFY-Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol: ◼</td>
</tr>
<tr>
<td>Purpose: Change attributes of process elements</td>
</tr>
<tr>
<td>Parameters:</td>
</tr>
<tr>
<td>- Element ID</td>
</tr>
<tr>
<td>- Attribute name</td>
</tr>
<tr>
<td>- Value to be assigned</td>
</tr>
</tbody>
</table>

Together with the option constraints, which are shown in Table 5, the Provop framework for modeling and managing large collections of business process variants is formed.

<table>
<thead>
<tr>
<th>Constraints Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implication</td>
<td>If different options shall be applied conjointly to the base process (e.g. due to semantically dependencies) the designer may explicitly define an implication constraint between them. Implication constraints are always directed.</td>
</tr>
<tr>
<td>Mutual Exclusion</td>
<td>It is helpful to describe which options must never be used in conjunction with each other when configuring a specific process variant. Provop visualizes the mutual exclusion constraint as bi-directed arrow between options.</td>
</tr>
</tbody>
</table>
Order of Application  
Options always have to be applied in sequence to a corresponding base process. As default Provop supports a low-level ordering mechanism based on time-stamps of options. As users may also define non-commutative options, however, this low-level ordering is enhanced by the provision of an explicit ordering constraint.

Hierarchy  
The definition of a hierarchy constraint between options allows combining the constraint type’s implication and application order. If an option is selected to configure a particular process variant and it has an ancestor in the option hierarchy, the change operations defined in the ancestor options will be applied as well. Thereby, a parental option is always applied before its child options. By introducing such hierarchy constraints, the number of constraints to be defined can be reduced.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>At least n out of m*</td>
<td>Each variant of the process family has been created by applying at least n options out of m specified ones.</td>
</tr>
<tr>
<td>At most n out of m*</td>
<td>Each variant of the process family has been created by applying at most n options out of m specified ones.</td>
</tr>
<tr>
<td>Exactly n out of m*</td>
<td>Each variant of the process family has been created by applying exactly n options out of m specified ones.</td>
</tr>
</tbody>
</table>

Table 5 - Provop options constraints

2.2.1.3 OVM-SaaS framework
In OVM-SaaS (Orthogonal Variability Model – Software as a service) approach [16], the key idea to address the management of variability in SaaS applications is to explicitly model the variability both customer-driven and realization-driven, the explanation of which are shown in Table 6. In addition, the binding of the SaaS variability is not solely dependent on the choices made by a consumer, but also on the choices made by the existing consumers. Thus, external variability and internal variability from the concept of software product line engineering are introduced. External variability is defined as the variability that can be communicated to the customer of the product line, whereas internal variability is only visible to the developers within the product line. In the SaaS context, customer-driven variability correlates with external variability and realization-driven variability nicely match the notion of internal variability.

<table>
<thead>
<tr>
<th>Source of variability</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer-driven (External)</td>
<td>Requirements may very among different consumers to the application. These differences in requirements require that the SaaS applications are customizable to capture for the varying requirements.</td>
</tr>
<tr>
<td>Realization-driven (Internal)</td>
<td>This is an alternative source to drive the introduction of a further level of variation to the SaaS application. As a running example, the “E-mail” feature of a SaaS application can be instantiate on a single infrastructure, which is shared but separately deployed between all consumers.</td>
</tr>
</tbody>
</table>

Table 6 - Source of variability
### 2.2.1.4 Summary
As shown in Table 7, the I- and PI-options framework aims for filling the technological gap in the support of cross-organizational collaborations. In the meanwhile, the Provop framework is designed to model and manage process variants. Moreover, the OVM-SaaS framework makes clear distinction between external and internal variability, which is derived from software product line engineering concept, to maintain and manage the variability. These three frameworks all offer the customers to customize their own business process via customizing business process variants (also known as behavior customization). From the consumer perspective, both frameworks offer high degree flexibility in the service execution. From the service provider perspective, by offering such high flexible service, the provider organization has an additional advantage to distinguish itself from other service providers. This, in return, improves the degree of efficiency and effectiveness of the cooperation between the information systems of the consumer and provider.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-&amp;PI-options framework</td>
<td>Fill in the technological gap in the support of cross-organizational collaborations</td>
</tr>
<tr>
<td>Provop framework</td>
<td>Model and manage process variants</td>
</tr>
<tr>
<td>OVM-SaaS framework</td>
<td>Clear distinction between external and internal variability</td>
</tr>
</tbody>
</table>

#### 2.2.2 Quality of Service (QoS) customization
While for control flow we have solutions to allow process users customize their own processes, in the case of resources and QoS, we have mechanisms to provider customization, but nothing to provide MASS-customization (that is, allowing the user to choose directly what they want). This section and the following section will elaborate more in detail for the case QoS and resources customization.

Quality of service (QoS) is increasingly being used to describe a broad collection of attributes, ranging from performance-related metrics used in computer networking and telecommunication to other metrics related to reliability, timeliness and security [20]. Moreover, QoS is also widely applied to guarantee the services, ranging from communication services to higher level middleware services. To construct a service that provides a fixed set of QoS guarantees is not typically realistic and cost-effective since different consumers that use the service may have very different requirements. The customized QoS well addressed this problem, which provide the options for customer to personalize his or her own requirements.

#### 2.2.2.1 Three-granularity framework
The paper [17] proposes a framework that comprises service management at three different granularity levels for adaptive service management. The three levels are explained as follows: (1) the static resource management optimizes the allocation of services to computing resources. (2) The dynamic resource management [21] uses a fuzzy logic based controller to remedy exceptional situations at runtime. These static and dynamic resource
adaptations exercise control in the large at the service instance level. (3) The adaptive control of service level agreements (SLAs) demands request level management [22] that exercises control in the small by intelligently scheduling individual requests. These three levels are shown in Figure 7.

With this three-tier approach of static resource management, dynamic resource management, and request level SLA enforcement, the quality of the service can be improved and the administrative overhead can be reduced. Moreover, the reduction of the TCO can achieve either more users can be served with the existing hardware or less hardware is required initially. In addition, the synergies of the control in the large and the small yield an improved quality of the managed systems in terms of reliability and compliance in QoS field.

2.2.2.2 CQoS framework
The paper [18] proposes a single unified framework to provide customized QoS service across multiple middleware platform, called CQoS architecture. This architecture allows the QoS attributes of a distributed object system to be customized transparently to client and server applications. Figure 8 provides the high-level architecture of CQoS. The middleware platform may be CORBA or Java RMI, but as noted above, the same approach can be used for any platform that supports a request-reply interaction paradigm, including remote procedure call (RPC) systems.

QoS customization can be done in different system levels, ranging from below the middleware, as a modification to the middleware, to as a service built on top of the
Each alternative has its own advantages and disadvantages, including factors such as transparency, effectiveness and efficiency of the approach, and the ease of the implementing and customizing such service enhancements. In the case of CQoS, implementing on top of the middleware surpassed other strategies, as the ability to use the higher-level primitives provided by the middleware for locating objects and for performing inter-object communication. CQoS can be inserted transparently to both application objects and middleware without any change in both.

The novel architecture proposed in [18] provides separation of concerns between the algorithms that implement QoS attributes and the details of the underlying middleware platform. This is accomplished by dividing CQoS into two components, each of which addresses one of these concerns and provides the appropriate interfaces for interaction between the components.

![Figure 8 - High-level view of CQoS architecture](image_url)

### 2.2.2.3 Summary

As shown in Table 8, the adaptive service management framework [17] presents a novel automatic administration concept that provides comprehensive adaptive service management at different granularities. The architecture of AutoGlobe and its fuzzy controller framework is proposed for dynamic adaptations at runtime. This architecture provides an up-to-date view of the load situation of the system, which is used to generate actions by fuzzy controllers. In the meantime, the CQoS architecture [18] provides customized combinations of multiple QoS attributes across multiple middleware platforms. It is a single novel unified framework that separate the algorithm that implement QoS attributes with the details of the underlying middleware platform. Both frameworks provide so called QoS customization in different scenario within the business process customization field.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three-granularity framework</td>
<td>Provide a comprehensive adaptive service management at different granularities, different service management at different level</td>
</tr>
</tbody>
</table>
2.2.3 Resource customization
A common way to manage resources in the process models is to assign a role, a capability and/or an organizational group to each task. In several flowchart-like notations, such as UML activity diagrams [23] or BPMN, this association is encoded by means of swimlanes. Each task (or activity) is assigned to a swimlane which may represent a role or an organizational unit. An activity can only be performed by a resource that belongs to all the partitions associated to it. In (extended) EPCs [24], roles or organizational units can be in any combinations attached to tasks, without any semantic requirements. Extended EPCs (eEPCs) have been partially formalized for simulation purposes in [25].

2.2.3.1 Role-based framework
The paper [13] defines more sophisticated role-based resource modeling features than in eEPCs and layers configuration features on top of them. These features go beyond the ones in UML Ads and BPMN. The paper [13] compares the approach with zur Muhlen [26], with the emphasis on the resource allocation, Russell et al. [27], which proposes a set of resource patterns describing various types of associations between resources and tasks, Ferraiolo et al. [28], which outline a reference model of well-accepted mechanisms for role-based access control, and Bertino et al. [29], who formalize authorization constraints and discuss their specification and enforcement in workflow management systems.

The paper [13] chooses to use EPCs as a base notation to define variability mechanisms along the data and resource perspectives since EPCs has three beneficial advantages. First, EPCs are widely used for reference process modeling (cf. the SAP reference model). Secondly, although lacking a precise definition, EPCs provide basic features for associating data nodes and roles to tasks. Finally, this choice allows us to build on top of the existing formal definition of the C-EPC notation.

In the meanwhile, under some circumstances, it may not be allowed to freely set, since sometimes it may depend on the configuration of others. For instance, the dependencies between functions and roles, or objects and functions may exist. These kinds of requirements can be classified according to the type of relations as shown in Table 9. More complex requirements can be derived from the combination of any two or more of the constraints shown in the table.

| Single Node requirements | constrain the configuration of a single node, where no dependency exists on other nodes. Req1 (restriction of dimensions): the Picture cut associated to the Spotting session can only be specialized, but not restricted along its optionality dimension (i.e. it cannot be excluded as this the initial input to the process). Req2: the Editor in Sound design cannot be specialized to Video Editor and the Editor in Picture |
editing cannot be specialized to Sound Editor, as the competencies are different. Req3: the control-flow connector XOR (c14) cannot be set to the sequence starting with the event Changes required, as this would lead to skip the whole premixing phase.

<table>
<thead>
<tr>
<th>Connector–Connector requirements</th>
<th>constrain the configuration of two or more connectors. Req4 (among role and object connectors): the range restriction of the OR-join (c11) and of the OR-join (c12) must be the same, so that at run-time, the choice of the role(s) linked by the first connector is consistent with the choice of the object(s) linked by the second connector, for the Progress update.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function–Function requirements</td>
<td>constrain the configuration of two or more functions. Req5: an editing project must have at least Music design or Sound design, thus we cannot exclude both. Req6: Music premixing requires Music design as Sound premixing requires Sound design.</td>
</tr>
<tr>
<td>Role–Role requirements</td>
<td>constrain the configuration of two or more roles. Req7 (on S): the Producer in Music premixing must be specialized in the same way as the Producer in Sound premixing, since these two roles are covered by the same person(s). Req8 (on M): a Spotting session needs at least a Composer or a Sound Designer.</td>
</tr>
<tr>
<td>Object–Object requirements</td>
<td>constrain the configuration of two or more objects. Req9 (on S): the Picture cuts, the Edited pictures and the Deliverable must have the same specialization, to ensure a consistent propagation of the picture medium. Req10 (on S, U): the Picture cut in Picture editing is consumed if and only if it is specialized to Film. Req11 (on M): the exclusion of Dialog cues from Sound design implies the exclusion of Dialog tracks, since these are produced based on the cues.</td>
</tr>
<tr>
<td>Connector–Node requirements</td>
<td>constrain the configuration of connectors and configurable nodes. Req12: the exclusion of function Progress update implies the restriction of the XOR-split (c14) to the sequence starting with Design finished, as the repetition of the design phase depends on the result of the Progress update.</td>
</tr>
<tr>
<td>Function–Role requirements</td>
<td>constrain the configuration of functions and roles. Req13 (on M): Music design must be excluded if the Composer is excluded from this function. On the other hand, if Sound Designer is excluded from Sound Design, this function can still be performed by the Editor.</td>
</tr>
<tr>
<td>Function–Object requirements</td>
<td>constrain the configuration of functions and objects. Req14 (on M): Progress update cannot be excluded if Temp music file in Music design or Temp sound file in Sound design are included, since the files are produced to be used later by this function. Otherwise, if Progress update is set to optional, the files cannot be made mandatory.</td>
</tr>
<tr>
<td>Role–Object requirements</td>
<td>constrain the configuration of roles and objects. Req15 (on M, S, U): a Negcutter is only required if the project is edited and delivered on Film. Thus, if this role is mandatory, the entire Picture cuts and Edited pictures, as well as the Deliverable, must be specialized to Film. In this case the Picture cut associated to the Picture editing cannot be set to be used. Req16 (on M): if a Composer does not partake in the</td>
</tr>
</tbody>
</table>
Progress update, the Temp music file and the Music notes must be excluded as they are required by this role.

Table 9 - Constraints table

2.2.3.2 Objects, rules and roles based framework
The paper [19] proposes a framework for WFMS to customize based on objects, rules and roles. It mentions three different areas that can be customized within the WFMS framework. These three points are concluded in Table 10.

| Activity ordering | Activities which are part of a certain workflow have to be coordinated with respect to their execution order by means of activity ordering policies. The basic activity ordering policies which are supported can be classified along two orthogonal dimensions, namely kind of control structure and kind of dependency. Whereas the former dimension comprises sequencing, branching and joining, the latter defines the starting points of the successor activities depending on the end of the predecessor activity or the start thereof. By combining these two dimensions different activity ordering policies emerge. |
| Agent selection | Agent Selection. For each activity one or more agents have to be selected by means of agent selection policies. An example of an agent selection policy is that agents being on vacation are not allowed to be selected. The decision which agent(s) to select for a specific activity is based on information concerning both the history of and the documents processed by the actual workflow. |
| Worklist Management | Worklist Management. Similar to activity ordering policies coordinating the execution order of activities, worklist management policies coordinate the execution order of several activities inserted into the worklist of a single agent |

Table 10 - Resources to be customized

The paper [13] and [19] fill the gap that control-flow customization fail to capture the participating resources, business objects in the customization domain. The proposed frameworks model the roles demanded to perform activities and business objects employed in activities, and capture variability in roles and objects and identify their interplay.

\[ M, S \] where M, S and U stand for the optionality, specialization and usage dimension, respectively
2.3 Mass-customizer capabilities

According to the research by Fabrizio Salvador, Pablo Martin de Holan and Frank Piller[30], three common capabilities that will determine the fundamental ability of a company to customize its offerings are developed: (1) solution space development, i.e. understanding customer needs and options over which those are likely to diverge, (2) robust process design, i.e. design an infrastructure to offer such options, making sure that customization does not hinder the company’s ability to execute its processes in an efficient and effective way, and (3) choice navigation, i.e. support customer selection of options minimizing the burden of choice. The details about these three successful mass-customizer capabilities are shown in Table 11.

<table>
<thead>
<tr>
<th>CAPABILITY</th>
<th>APPROACHES TO DEVELOP CAPABILITIES</th>
</tr>
</thead>
</table>
| **Solution Space Development:** Identify the product attributes along which customer needs mostly diverge | • **Innovation toolkits:** Software that enables large pools of customers to translate their preferences into unique product variants, allowing each customer to highlight possibly unsatisfied needs.  
• **Virtual concept testing:** An approach for efficiently submitting scores of differentiated product concepts to prospective customers via virtual prototype creation and evaluation.  
• **Customers experience intelligence:** Tool for continuously collecting data on customer transactions, behaviors or experiences and analyzing that information to determine customer preferences. |
| **Robust Process Design:** Reuse or recombine existing organizational and value chain resources to fulfill a stream of differentiated customers’ needs | • **Flexible automation:** Automation that is not fixed or rigid and can handle the customization of tangible or intangible goods.  
• **Process modularity:** Segmenting existing organizational and value-chain resources into modules that can be reused or recombined to fulfill differentiated customers’ needs.  
• **Adaptive human capital:** Developing managers and employees who can deal with new and ambiguous tasks. |
| **Choice Navigation:** Support the customers in identifying their own solutions, while minimizing complexity and the burden of choice | • **Assortment matching:** Software that matches the characteristics of an existing solutions space (that is, a set of options) with a model of the customer’s needs and then makes product recommendations.  
• **Fast-cycle, trial-and-error learning:** An approach that empowers customers to build models of their own needs and interactively test the match between those models and the available solutions.  
• **Embedded configuration:** Products that “understand” how they should adapt to the customer and then reconfigure themselves accordingly. |

Table 11 - Three capabilities for successful mass-customizer [23]
2.4 Proposed business process monitoring customization infrastructure

Due to the development of information systems, especially in the commercial use, business process development undergoes a quick and unstoppable way. However, traditional business process management didn’t incorporate configuration or customization ideas, which make the pool of business process models extremely large and redundant. “Mass-customization” in business process management, which challenges the one-size-fits-all assumption of mass production, provides customers the opportunity to customize the way in which a process will be enacted.

Traditional way to customize, called “configuration”, in information systems is a design time concern, which aims at designing standard process across the implementing organization. While process standardization across the enterprise promotes uniformity and interoperability [10], it is also seen as a constraining institutional factor for large companies with diverse business units [31], limiting the flexibility of the company. Our research will focus on a much more dynamic perspective on process customization, which in information systems allows each single client to specify his or her own requirements at runtime.

Then we make some literature study on which aspects of business process can be customized. Business process customization can be ranged from behavior to QoS customization or data and resource customization. However, there is lack of literature about business process monitorability customization, which is the reason why we want to focus on.

Business process monitoring customization can be treated as a management aspect and the business entities need information about the activities taken places inside the business landscape (both internal units and external units) so that they can react to the changes quickly and in time. However, the literature about business process monitoring customization has not yet been extensively investigated based on our literature review.

Therefore, we decide to do some research based on business process monitoring customization. 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.

Fabrizio Salvador, Pablo Martin de Holan and Frank Piller [30] proposes three capabilities that will determine whether the mass-customizer will be successful or not. Our research focused on the provider side that designs the monitoring infrastructure, which aims at supporting capabilities (1) and (2). To support the provider in the identification of the solution space, in fact, a set of patterns for the conceptual design of mass-customizable business process monitoring infrastructures will be provided. The patterns are positioned in a framework providing a multidimensional classification space. The multidimensional space is constructed by reasoning on the literature for software monitoring, Web service, and business process monitoring and by using existing, context-specific solutions as verification and illustration techniques. To support the robust design of customized monitoring
infrastructures, our research will show how to combine patterns depending on specific contextual requirements and provide a general implementation about the business process monitoring customization.

A general structure of business process customization is shown in Figure 9, from the figure we can see each single client firstly interact with monitoring client panel and personalized their panel by selecting different options provided by the Monitoring infrastructure. Then the monitoring client will interact with monitoring infrastructure, which will trigger process-aware information systems to obtain required data. As soon as the data is sent to Monitoring infrastructure, a customized panel will be instantiated with the data to the target client.

Figure 9 - Structure of business process customizer
After a careful literature review on business process customization, the focus narrows into the management aspect that has not yet been extensively investigated, that is business process monitoring and mass customization. Generally speaking, business entities need information about the activities taken places within the business landscape (at internal units, partner entities or third parties) so that they can properly react and adapt to them. Thus, business entities need mechanisms that ensure the collection of relevant data from the business environment (also known as [32]). In B2B, customers may use monitoring information, for example, to synchronize their own internal processes or to assess the provider’s behavior, especially in dynamic relationships where customers and providers are often not likely to have conducted business together in the past. In this context, different customer is likely to have different specific monitoring requirement. Customization of monitoring requires the 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 whole process.

According to Salvador et al. 2009, a successful mass-customizer should develop three capabilities [30] (see Table 11): (a) solution space development, i.e. understanding customer needs and options over which those are likely to diverge, (b) robust process design, i.e. design an infrastructure to offer such options, making sure that customization does not hinder the company’s ability to execute its processes in an efficient and effective way, and (c) choice navigation, i.e. support customer selection of options minimizing the burden of choice.

Focusing on the provider designing the monitoring infrastructure, we aim at supporting the capabilities (a) and (b). To support the provider in the identification of the solution space, in fact, we provide a set of patterns for the conceptual design of mass-customizable business process monitoring infrastructures. The patterns are positioned in a framework providing a multidimensional classification space. The multidimensional space is constructed by reasoning on the literature for software monitoring, Web service, and business process monitoring and by using existing, context-specific solutions as verification and illustration techniques. To support the robust design of customized monitoring infrastructures, we show how to combine patterns depending on specific contextual requirements using an example in online advertising.

In order to design a monitoring infrastructure, which supports business process monitorability customization, the mass-customization capabilities discussed in chapter 2 are
tailored as shown in Table 12. Table 12 focuses on the provider side, by providing a set of patterns and multidimensional classification spaces, together with the necessary combination of the patterns, the capabilities “solution space development” and “robust process design” are achieved. For future research reference, choice navigation is also shown in Table 12.

<table>
<thead>
<tr>
<th>CAPABILITY</th>
<th>APPROACHES TO DEVELOP CAPABILITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution Space Development:</td>
<td>• <strong>Set of patterns</strong>: A set of patterns for the conceptual design of mass-customizable business process monitoring infrastructures.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Multidimensional classification</strong>: Positioned patterns in the framework support multidimensional classification space.</td>
</tr>
<tr>
<td>Robust Process Design:</td>
<td>• <strong>Combine patterns</strong>: Patterns are combined according to specific contextual requirements</td>
</tr>
<tr>
<td>Choice Navigation:</td>
<td>• <strong>Minimize burden of choices</strong>: A set of options to support customers to minimize the burden of choices</td>
</tr>
</tbody>
</table>

Table 12 – Capabilities for successful monitorability mass-customizer

3.1 Monitoring dimensions
In Figure 11 (a), we sketch the general outline of the monitoring architecture, which is conceptual miniature of Figure 9. In the business process provider domain, a monitoring infrastructure (also called sensor or observer [33][34]) capturing relevant information is built around a business process engine. A business process engine is meant in a broad sense - it is a component producing process information (activity or process states, data values, etc.). Although the monitoring infrastructure can be built in (or even be an integral part of) the business process engine [33], conceptually, it is a separate entity [34]. Considering it as a separate entity promotes separation of concerns with respect to the underlying process engine, leading to a more focused and context-independent analysis of the monitoring infrastructure component and its possible customization. In the customer (also called controller [33][34]) domain, a monitoring client obtains the information captured by the monitoring infrastructure. It processes the information obtained and if desired exerts control over the business process engine.

In many scenarios, the monitoring client can be intermediary for the actual business entity that requires the monitoring information. Monitoring may take place within the organization, e.g. between different but independent business units, or in a cross-organizational settings, e.g. between two different companies. In order to capture such information, a monitoring infrastructure is built around the business process engine, whereas in the meantime, the monitoring infrastructure is a separate entity. In the client domain, a monitoring client obtains the information captured by the monitoring infrastructure. It process the information obtained and if desired exerts control over the business process engine.
The dimensions of our framework identify what parts of the scenario in Figure 11 (a) can be customized by the monitoring client, i.e. they define the monitoring solution space for the customer of the business process. The first two dimensions concern the context in which monitoring has to occur. In particular, we consider the monitoring variable and the anchoring points, defined as shown in Table 13.

<table>
<thead>
<tr>
<th>Monitoring Variable (MV)</th>
<th>The MV specifies the object of monitoring, i.e. the process information that the Monitoring Infrastructure (MI) obtains from the Business Process Engine (BPE) and makes available to the Monitoring Client (MC). It has a domain, which specifies the (range of) values that it can assume, and a unit of measure, if needed. In the world of software programs monitoring, the monitoring variable is the target, for instance, of a watch for debugging. In business process monitoring, the monitoring variable may range from infrastructure-level data, such as timestamps of service calls [35], to application-level process data, such as the status of an activity or domain specific data produced by an activity [36]. Note that a value of the monitoring variable represents a single data element captured by MI during the execution of the business process, e.g. the timestamp of an order, the warehouse level at a specific point in time, the unique id of a user executing a specific activity. Captured values can be stored by MI during the execution of a business process and made available in batches to MC. A classification of possible MVs is out of scope in this paper.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anchoring Point (AP)</td>
<td>The MI is enabled within a specific scope of the process to be monitored. Anchoring points specify the scope of the process within which the MI is enabled. The notion of anchoring point derives from the literature on software program monitoring, where running the monitoring program in the same memory space as the monitored program can be costly and, therefore, the monitoring program has to be enabled only when strictly necessary [37]. In a business setting, while in many cases we can make the hypothesis that monitoring is permanently enabled, defining anchoring points may be helpful when capturing and making available the values of MV to MC is very costly. Intrusive process monitoring [35] is an example of this scenario, since the execution of monitoring statements blocks and, therefore, delays, the execution of the process. In such a scenario, the MC may want to enable monitoring only when strictly necessary. In a business setting, while in many cases we can make the hypothesis that monitoring is permanently enabled, defining anchoring points may be helpful when capturing and making available the values of MV to MC is very costly. Intrusive process monitoring [35] is an example of this scenario, since the execution of monitoring statements blocks and, therefore, delays, the execution of the process. In such a scenario, the MC may want to enable monitoring only when strictly necessary. In the remainder, we refer</td>
</tr>
</tbody>
</table>
The conceptual outline of the monitoring solution is refined from Table 13, and the lifecycle of communications among different elements of the architecture which are shown in Figure 11 (b). From Figure 11 we can see that the first phase is commanding the acquisition of monitoring data. This can be done either by MC or by MI (Phase 1). For instance, MC may specify that MI has to acquire values of a specific MV periodically or may want to be allowed to command the acquisition pro-actively. The second phase concerns MI obtaining the required data from BPE (Phase 2). Since this phase is internal to the business process provider domain and cannot be customized, we leave it out of the scope of this paper. Then, MI may have to manage, e.g. store in batches, the monitoring data obtained by BPE (Phase 3) before supplying them to MC (Phase 4). Eventually, MC processes monitoring data and may decide to exert control on the monitored process executed by BPE (Phase 5). This last phase is also out of scope in this paper. Phases 1, 3, and 4 of the lifecycle in Figure 11 (b) are characterized by one monitoring dimension in the framework, since they involve aspects that are customizable by MC. Each dimension has a set of options. Options represent the solution space for the customization of process monitoring, that is, a customized monitoring infrastructure can be built once the customer has chosen one option for each possible monitoring dimension. The monitoring dimensions and their possible options are illustrated in Table 14.
before supplying them to MC. New values obtained from BPE can rewrite old values captured for the same MV or the obtained values can be stored (persisted), e.g. to produce historical series of values of MV. When supplied to MC, values can be consumed, i.e. they will not be available in the future to MC, or they can just be read, remaining available also in the future. Thus, we identify four options for MI ↔, i.e. (1) rewrite-consume, (2) rewrite-read, (3) persist-consume, and (4) persist-read. Moreover, we extend two more options for MI ↔, i.e. (1) periodic and (2) threshold. These two options can also be applied to the Supply Data to MC dimension.

| Supply Data to MC (MI-MC) | This phase is characterized by the dimension MI-MC↔ referring to the direction of the interaction between MI and MC. For MI-MC↔ we define the options push and pull. The option push models cases in which MI pushes monitoring values to MC, whereas pull models cases in which MI sends values of MV only after having received a request from MC. Generally, communication between MI and MC is a distributed systems communication issue and its customization may require the definition of additional dimensions, such as space and time decoupling option [38], [39]. However, since those aspects are not monitoring-specific, we do not further discuss them here. |

### Table 14 - Monitoring options

#### 3.2 Monitoring Patterns

In this section, we use the monitoring dimensions and present the design of internal data and control flows of the Monitoring Infrastructure (MI). This is required for the provider to offer mass-customization monitoring capabilities to its customers. Following common principles in the design of distributed systems [34], we first define the interfaces required between MI and MC, and MI and BPE, respectively, to support the monitoring dimensions. Then, we propose a modeling pattern for each option characterizing the monitoring dimensions. Modeling patterns specify the data and control flow of MI’s internal implementation of the interfaces.

For business process modeling purposes, we use Colored Petri Nets (CPNs) [39][40]. CPNs have been chosen because they have a graphical representation, they have mature and freely available tool support, e.g. CPN Tools, and they have a precise semantic that can be translated to or directly implemented in other languages, e.g. Event-driven Process Chains [24] or YAWL [36], used by commercial and non-commercial workflow engines. Readers unfamiliar with CPNs’ notation are referred to [40]. We use two token colors in our patterns, i.e. UNIT and MV. UNIT represents the default color of black tokens, and it is used to model the control flow in our modeling patterns. Tokens of color MV represent a single value of the monitoring variable MV. The precise definition of the color MV, e.g. possible values and unit of measure, is part of the definition of the monitoring context and we do not further discuss it in this paper.
Figure 12 shows the interfaces between the elements of the monitoring solution MI, MC, and BPE. Note that, although aspects related to the interaction between MC and BPE are internal to the provider domain, and therefore not subject to possible customization, considering the interfaces between MI and BPE allows separation of concerns between business process execution and monitoring and, therefore, for more robust patterns for customizable process monitoring infrastructures.

MI requires one interface to receive acquisition triggers from MC (MC-ACQ-REQ) and supply data to MC (SUPPLY). Optionally, MI may require also an interface to receive supply requests from MC (SUPPLY REQ). This interface is necessary only when MC decides to pull monitoring data from MI. The interface between MI and BPE is constituted by a generic interface for receiving monitoring values (OBTAIN), by the two anchoring points, and by interface BPE-ACQ-REQ allowing MI to command acquisition of values of MV by BPE. Note that Figure 12 does not show the color of tokens in places connecting interfaces. As shown
later, the color of such places is determined by the modeling pattern realizing the MI’s internal implementation of the corresponding interface(s).

Concerning the context, MI is required to expose interfaces to define the anchoring points of monitoring to the business process executed by BPE. The tokens required to fire the transitions AP-START and AP-END are produced by BPE when the process execution reaches the point enabling and disabling the monitoring, respectively. Figure 13 shows the pattern for the implementation within MI of the anchoring point business logic. Specifically, the generic interface <INT>, which can be any of the ones defined in Figure 12, becomes enabled after the firing of the AP-START transition and is no longer enabled after the AP-END transition has fired.

Figure 13 - Pattern for anchoring point implementation

![Figure 13 - Pattern for anchoring point implementation](image)

Monitoring Trigger (μ ▶). The patterns modeling the options of dimension μ ▶ ? implement the ACQUIRE-REC and BPE-ACQ-REQ interfaces of MI.

The patterns corresponding to the three values MI-trg, MC-trg, and mix-trg are shown in Figure 14. In Figure 14(a), the transition INTERNAL TRIGGER captures MI’s business logic to trigger acquisition, e.g. periodically or on change, which can be customized by MC. In Figure
the acquisition is commanded by MC, by firing the transition ACQUIRE REQ, whereas Figure 14(c) shows the mix case, i.e. MC or MI can both trigger an acquisition request.

**Supply data to MC (MI-MC).** The patterns for the push and pull options of the dimension MI-MC←→ are shown in Figure 15 (a) and Figure 15 (b), respectively. The push option requires MI to only expose the SUPPLY interface. The transition SUPPLY TRIGGER fires accordingly to the logic chosen by the customer to receive monitoring values. The customer, for instance, may require monitoring values periodically or only when values exceed a certain threshold. For the pull option, the SUPPLY interface can fire, i.e. supply a monitoring value, only after a request is received. Note that that this implemented by limiting the control exerted by the anchoring point to the SUPPLY-REQ interface.

**Manage data (MI).** The basic patterns for the MI dimension implement the connection between the interfaces SUPPLY and OBTAIN exposed by MI. Figure 16 shows the patterns for the four options.

Note that the place bpe stores values of MV ready to be obtained by MI, whereas the place mc stores the value or set of values supplied to the monitoring client MC. Also, note that the color L_MV represents a list of tokens of color MV.

In the rewrite- options, the new value of MV v_new always replaces the old value v_old. In the persist- options, the values of MV are stored by MI in a list lv (the list in this case represents a generic data structure); a new monitoring value v_new is simply inserted into lv.

In the -consume options, the SUPPLY transition, when fired, replaces the current monitoring value stored by MI with the default, whereas in the -read options the monitoring value is put back in the place mi_db after being read and can be read again in the future by MC. Note that the default monitoring value is the empty list [] for the persist- options and the token with value “dft” of color MV for the rewrite- options.
Moreover, besides the basic patterns, we also extend some more patterns for the MI dimension. Figure 17 shows the patterns for the two options.

In the periodic- options, after a fixed time period, the S/O transition will be triggered. (In the CPN example, the time unit is set to be 4; in practice, it can be any time units.) In the threshold- options, when the S/O transition is triggered, only those which can meet the requirements (in the CPN example, the threshold is 4; in practice, it can be greater than, equal or less than 4.) will be supplied or obtained.

Note that the name of the transition as shown in Figure 17 is S/O, which means it can be either supply or obtain. These two patterns can be applied to all communications between BPE-MI and MI-MC, which means that these two patterns would be the options for the MI dimension and the MI-MC dimension.
3.3 Combining patterns to design Monitoring infrastructure

To demonstrate the value of the framework, we provide an example for its application in an on-line advertising scenario. The example shows how the framework can be applied for the design of a customizable monitoring solution and the advantages that it offers to the designers of the monitoring infrastructure at the provider side.

The advertising provider (e.g. a newspaper) offers advertising space to companies (customers). In a simplified scenario (see Figure 18), the customer sends the advertisement to the provider and when the time agreed to start the campaign comes, the provider starts the advertising campaign by publishing the ad in its newspaper. In a basic scenario, the ad is shown each time a reader loads the newspaper page, while in a more complex case, the most relevant and highest priced ad is shown in a specific advertising spot (e.g. in Google AdWords). When the budget of the customer is depleted and/or the number of agreed appearances of the ad is reached, the campaign ends. The customer may ask to change its campaign if it observes that the ad is not reaching the target audience, the campaign has little impact, etc. This business case is described in greater detail in [41].

In a traditional advertising setting, the provider offers a fixed set of tools to all customers for monitoring their campaigns. For example, the customers can monitor the IP addresses of the readers seeing their ad, the number of shows of their ad, number of clicks on their ad,
the spots where the ad has been published (banner, column, pop-up, in-text), etc. Typically, customers have to access their account to view this information. This monitoring advertising scheme does not address the individual preferences of the customers. Customers may be interested in receiving the monitoring information directly instead of having to query it from the provider; they may be interested in obtaining the information in an aggregated format at the end of the campaign or be constantly updated to be able to adapt their campaign; they may be interested, motivated by a cheaper price, in obtaining only some of the monitoring information instead of monitoring all possible variables, etc. Next, we describe how our framework can be applied to set up a mass-customizable monitoring infrastructure. For brevity, we focus on the construction of monitoring infrastructure for only one monitoring variable, i.e., the number of clicks on an ad, which is the most straightforward indicator of the ad success and profitability over time.

To apply the framework for the mass-customization of a MV, the provider has to consider the possible options from the monitoring dimensions presented in section 3.1 and section 3.2 to offer to its customers (see Table 15). The MV in this case is the current value of number of clicks on the ad of the customer. We hypothesize that the provider (newspaper), through an advertising engine, keeps track continuously of this value. The customer wants to monitor MV along the campaign. Therefore, the anchoring point enabling and disabling monitoring are the start and the end of the campaign, respectively. Note that, in principles, customers may choose a different anchoring point, for instance enabling monitoring only after the campaign has been changed a first time.

<table>
<thead>
<tr>
<th>Monitoring Variable</th>
<th>Number of clicks on the customer’s ad (cumulative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anchoring Point</td>
<td>“Start Campaign” (enabling); “End Campaign” (disabling)</td>
</tr>
<tr>
<td>Monitoring Trigger</td>
<td>$\mu \uparrow?:$ MC-trg, MI-trg</td>
</tr>
<tr>
<td>Manage Data</td>
<td>MI₪: persist-read, persist-consume</td>
</tr>
<tr>
<td>Supply Data to MC</td>
<td>MI-MC $\leftrightarrow$ push, pull (with periodic or threshold)</td>
</tr>
</tbody>
</table>

Table 15 - Possible monitoring dimension values for the IP address MV

A customer may like to be pushed monitoring information or to pull it (thus, MI-MC$\leftrightarrow$=push or MI-MC$\leftrightarrow$=pull). The customer may prefer to monitor the MV only at a specific point in time that cannot be revealed (e.g., it is not known) to the provider (hence, $\mu \uparrow?:$MC-trg) but may also like to delegate the acquisition of the monitoring values to the provider at a pre-agreed time, periodically or when the number of clicks exceeds a certain threshold (hence, $\mu \uparrow?:$MI-trg). Typically, customers would prefer to have all the monitored values stored by the provider, so that these can be queried any time later on and used to analyze the number of clicks trend over time (hence, MI₪=persist-read). Of course, if storage space is crucial for the provider, it may offer some incentives (e.g., financial) to the customers to choose also MI₪=persist-consume.
Thus, having the framework as a guiding tool in the set of possible monitoring styles, the provider has straightforwardly defined all possible monitoring styles for the number of clicks MV. Each customer interested in monitoring information on the cumulative number of clicks is presented with a set of possible options. Table 16 presents two possible sets of choices for customers A and B. Customer A delegates the acquisition of monitoring values to the provider at the beginning of its campaign, e.g. every six hours, it wants to be able to pull monitoring information when needed, and it also requires that the information is kept after it reads it to be able, for instance, to show the trend of this MV over time as soon as the information is pulled from MI. Thus, each time Customer A pulls monitoring information it will get a list of sampled (one every six hours) values of the number of clicks on its ad since the publication of the ad. Customer B wants to be able to specify when the values of number of clicks have to be acquired (μ▶?=MC-trg) and to automatically receive the monitoring information, for instance as soon as this is acquired by the provider or at the end of each day (MI-MC↔: push; where the trigger for the push is the availability of a new value for number of clicks MV or the end of the day). Customer B is only interested in the values that contains more than 2 records (in the CPN example as shown in Figure 20, the “length(lv)>2” function accomplish this), which is considered as a threshold. Moreover, Customer B also does not require the provider to locally store the acquired values (MI₪: persist-consume). This is a reasonable assumption, for instance, when data are directly pushed to MC as soon as they are acquired by the provider.
As different customers may choose different values in each dimension (as demonstrated in Table 16), the provider has to be prepared to support each possible combination. Having the patterns corresponding for each monitoring dimension values pre-defined in the framework, the provider can now directly apply them for these specific MV and AP. In Figure 19, we demonstrate this, by showing the customized MI for Customer A. Note that Figure 19 includes also a possible characterization for the internal implementation of BPE. The customized MI to satisfy the requirements of Customer B can be similarly derived by combing the requirements specified in Table 16 (see Figure 20).

To summarize, the framework has given the set of possible values to the provider to identify all possible options valid for this specific MV (i.e. the solution space). Again using the framework, the provider can directly apply pre-defined patterns to ensure support for these monitoring styles whenever they are selected by a specific customer (i.e. achieving a robust process supporting customization).
4 Implementation and Architecture

In this chapter, we will show the implementation and the architecture of the business process mass-customization. Section 4.1 provides the implementation of the monitoring dimensions and patterns, which is only a simple proof of the concept of the feasibility. Then Section 4.2 describes the design of a more complex architecture using state-of-the-art technology, such as web services, ESB (enterprise service bus) etc. Finally, in section 4.3, the software quality evaluation criteria and the corresponding pros and cons analysis are proposed.

4.1 Implementation

All the implementations are in Netbeans IDE 7.1 with the programming language JAVA. Section 4.1.1 shows the class diagram describing the elements of the implementation. Section 4.1.2 shows the interface and how the program runs and looks like. Section 4.1.3 explains how the dimensions and patterns are implemented and look like in the demo. Section 4.1.4 does a brief analysis of the implementation with the pros and cons.

4.1.1 Class Diagram

The implementation uses the observer pattern [42] (see Figure 22), which is a kind of software design pattern. There are two main components in this pattern, namely the object and the observers. The object, also known as the subject, maintains a list of its dependents, called observers. The observer will be notified automatically by any state change, usually by calling one of the methods in the subject.
In essence, the observer pattern is to “Define a one-to-many dependency between objects so that once one object changes state, all its dependents are notified and updated automatically.” Using the observer pattern, the main class diagram for the java implementation is shown in Figure 23.
The classes or interfaces in this diagram are as follows:

**Subject**: This is an interface and it defines all the possible methods for the classes that may implement it. There are two general methods for all the observer patterns, they are register and unregister functions. Besides these, there are three methods, switchPushPullPattern, switchRewritePersistPattern and switch ConsumeReadPattern. They are used to switch between push and pull, rewrite and persist and consume and read, respectively.
Publisher: This is a class that implements the interface subject. It maintains two user lists, pushUserList and pullUserList, which are used to manage users who choose the push pattern option and pull pattern option, respectively.

Observer: This is an interface and it defines all the possible methods for the classes that would like to receive information from the publisher. The update method is similar to the notify method in the observer pattern, it will notify all the subscribers once there are updates. Moreover, it also provides the get and set methods to manage the push/pull, rewrite/persist and consume/read flags.

Subscriber: This is a class that implements the interface observer. Besides the implementation of the methods in Observer, it also provides a constructor function; each time, the new subscriber can initialize itself with that function and register in the publisher list.

4.1.2 Running the program
If we run the program, there will be a Log in window as shown in Figure 24. After typing the correct username and password, the user is able to login to the system. The login function ensures that the generated monitoring interface is only for this specific user.

![Log In](image)

Figure 24 - Log in

Then, a monitoring variables selection screen is shown in Figure 25. There are five monitoring variables from A to E, which should be consistent with the business processes the user wants to monitor. If the user wants to monitor variable C only, he or she will select C, which is shown in Figure 26.
After the selection of the monitoring variables in the previous step, the user is able to enter into the pattern choice navigation screen as shown in Figure 27. The user can choose between push and pull, rewrite and persist, consume and read as well as the definition of the threshold.
Suppose the user select the pattern “push-persist-read-30”, the result block is shown in Figure 28. Each time, as soon as the new value for variable C comes, it will immediately push the value to the user. Because of the “persist” and “read” pattern, the old data will be displayed together with the new arrival data.

4.1.3 Pattern explanation

In order to explain all the monitoring patterns, we provide a full list of how we implement these in JAVA. As shown in Figure 29, the left side shows the entire possible push pattern, within this category, each time, when there is new arrival value; it will immediately push to the user. However, the right side show the entire possible pull pattern, the difference relies on the fact that it requires the user to get the values.
If the user selects the “push” pattern to supply data from MI to MC, the patterns to manage data vary. From the top to bottom are “Persist-Read”, “Persist-Consume”, “Rewrite-Read”, “Rewrite-Consumed” patterns. If the user selects the “pull” pattern for the “Supply data from MI to MC” dimension, from the top to bottom, as shown in the right part of Figure 29, are “Persist-Read”, “Persist-Consumed”, and “Rewrite-Read”, “Rewrite-Consumed”.

4.1.4 Analysis of the implementation
In practice, there is possibility to accomplish all the monitoring dimensions and monitoring patterns in JAVA language. However, the implementation only provides a practical proof of the concept and it still lacks of the interaction with the business process engine. In order to provide a whole view of the business process monitoring customization infrastructure, a more complete architecture is proposed in Section 4.2. (This will be elaborated in detail later) The communications between different components are via WSDL and technically put,
it’s feasible to translate from JAVA to WSDL. As a result, the feasibility of the implementation of the monitoring dimensions and patterns in plain JAVA implies that the proposed business process monitoring customization infrastructure in in section 4.2 is proven to be practically feasible.
4.2 Architecture

Section 4.1 already showed that it is feasible to implement the monitoring dimensions and patterns in Java. In this section, we want to make our mass-customization monitorability support to be more generic, i.e. besides BPEL engine, it can be also built on top of several different business process engine, as shown in Figure 30. In order to provide a universal architecture which support different BPEL (Business Process Execution Language) engine, as well as different business process models, and different monitorability customization requirements from clients, the middleware is proposed in between. As defined in [43], middleware is software which is ‘in the middle’ between different software modules. Technically put, middleware is a software platform designed to effectively and efficiently facilitate the interconnection of large set of software modules in a complex environment. As shown in Figure 30, the middleware will act as a bridge between BPEL engine and the monitoring clients.

![Figure 30 - Architecture](image)

Enterprise service bus (ESB), which served as middleware (see Figure 30), is a software architecture model used for designing and implementing the interaction and communication between mutually interacting software applications in Service Oriented Architecture. [44] Since the ESB translates the message into the correct type and to the correct service, BPEL and Java, as the supported service, can communicate in the same way with the ESB. The following subsections are described as follows: Section 4.2.1 describes the basic concepts about BPEL and how the BPEL interact with other technology as service provider on one hand, and with end-clients on the other. Section 4.2.2 describes the architecture of YAWL system and how the YAWL system uses WSDL as its internal interaction with other YAWL components and with external exposure to the clients. Section 4.2.3 does a literature study about how the other business process languages mapped into BPEL in a theoretical level. Section 4.2.5 analyzed the pros and cons of the architecture.
4.2.1 Business Process Execution Language for Web Services (BPEL)

Business Process Execution Language (BPEL), short for Web Services Business Process Execution Language (WS-BPEL) is an OASIS standard executable language for specifying actions within business processes with web services. As indicated in [45], BPEL provides the specification language and intends to support the modeling for two types of processes: executable and abstract business processes. Executable business processes model actual behavior of a participant in a business interaction and specify the execution order between a number of activities constituting the process, the partners involved in the process, the messages exchanged between these partners, and the fault and exception handling specifying the behavior in cases of errors and exceptions. An abstract business process can be treated as a business protocol and may hide some of the required concrete operational details. It serves a descriptive role and specifies the message exchange behavior between different parties without revealing the internal behavior.

In BPEL, web services are described in Web Services Description Language (WSDL), which is an XML-based language that is used for describing the functionality offered by a Web service. Meanwhile, BPEL provides standard-based orchestration of these operations with multiple web services in the right order to perform a business process. In essence, a BPEL process itself is kind of flow-chart, where each element in the process is an activity. There are two kinds of activities in BPEL, basic activity and structured activity. The set of basic activities consist <invoke>, invoke a one-way request/response operation on a portType offered by a partner; <receive>, do a blocking wait for a matching message to arrive and can be the instantiator of the business process; <reply>, send a message in reply to a message that was received through <receive> ; <wait>, waiting for some time; <assign>, copying data from one place to another; <throw>, indicating errors in the execution; <terminate>, terminating the entire service instance; and <empty>, doing nothing.

In the meantime, the set of structured activities contain <sequence>, perform activities in a sequential order; <flow>, perform activities in parallel; <if>, conditional choice of activities; <scope>, enclose multiple activities in a single scope; <switch>, for conditional routing; <while>, for looping; <pick>, for race conditions based on timing or external triggers; (For the complete activities in BPEL, please check Table 17. A detailed or complete description of BPEL is out-of-the-scope of this paper. For more details, the reader is referred to [45]). Structured activities can be nested and combined in arbitrary way, which enable the presentation of the complex structures.

<table>
<thead>
<tr>
<th>Basic Activities</th>
<th>Structured Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;invoke&gt;</td>
<td>&lt;if&gt;</td>
</tr>
<tr>
<td>&lt;receive&gt;</td>
<td>&lt;while&gt;</td>
</tr>
<tr>
<td>&lt;reply&gt;</td>
<td>&lt;repeatUntil&gt;</td>
</tr>
<tr>
<td>&lt;assign&gt;</td>
<td>&lt;foreach&gt;</td>
</tr>
</tbody>
</table>

The reason why we introduce BPEL into our project relies on the fact that BPEL not only is currently the most widespread language for composing Web services but also itself can define business processes that interact with external entities through web service operations defined with WSDL. The interaction can be "abstract" in the sense that the focus is on portType definitions instead of port definitions. The usage of internal business processes is executable while the external business process is abstract. As shown in Figure 31, each service container, such as J2EE, XSLT, and JMS, registers the services they provide via WSDL to the Normalized Message Router (NMR). As the central message delivery mechanism, NMR delivers normalized messages to each service container. Service containers can host any services, located locally or remotely. This architecture enables the BPEL engine to utilize different services described in JMS, XSLT, and etc.

4.2.2 Yet Another Workflow Language (YAWL)

YAWL (Yet Another Workflow Language) is based on a rigorous analysis of existing workflow management systems and related standards using a comprehensive set of workflow patterns.[46] Compared to Petri Nets, YAWL overcomes the limitations of Petri Nets, YAWL has been extended with features to support patterns involving multiple instances, advanced synchronization patterns, and cancellation patterns. Moreover, YAWL allows for hierarchical decomposition and can handle arbitrarily complex data.
In the YAWL system, all end-users, applications and organizations are abstracted as so-called YAWL services. There are four kinds of YAWL services, as shown in Figure 32, (1) YAWL worklist handler, (2) YAWL web services broker, (3) YAWL interoperability broker, and (4) custom YAWL services. Compared to traditional worklist handler, which is embedded in the workflow engine, YAWL worklist handler is decoupled from the engine. The YAWL web services broker acts as the glue between the engine and other web services. This design enables the YAWL engine to interact with different kinds of web services. Accordingly, the YAWL web services broker acts as the mediator between the YAWL engine and external web services that may be invoked by the engine to delegate tasks. The YAWL interoperability broker is a service designed to interconnect different workflow engines. [46] For instance, a custom YAWL service, which could offer communication with printers, mobile devices, assembly robots, etc., could be subcontracted to another system where the service still corresponds to the whole process. Note that it is also possible that there are multiple services of the same type, e.g. multiple worklist handlers, web services brokers, and interoperability brokers. For example, there may exist multiple implementations of worklist services.
handlers (e.g., customized for a specific application domain or organization) and the same worklist handler may be instantiated multiple times (e.g., one worklist handler per geographical region).

Workflow specifications are managed by the YAWL repository and the case data (workflow instance) are managed directly by the YAWL engine. Clearly, it is also very crucial to have a management tool that can be used to control over the workflow instances manually (e.g. modify a case data or a workflow specification). The role of the YAWL manager should also consist of providing information about the state of running workflow instances, and details or aggregated data about completed instances.

From the middle part of Figure 32, the YAWL engine has two groups of interfaces: (A) interfaces capturing the communications between the YAWL designer and the YAWL manager on the one side and the YAWL engine on the other side; and (B) interfaces capturing the communications between all kinds of YAWL services and the YAWL engine. If we look at the reference model of the workflow Management Coalition (WfMC) [47] (See in Figure 33), the group of interfaces (A) can be matched with Interface 1 (Process Definition tools) and Interface 5 (Administration and Monitoring tools) of WfMC. The group of interfaces (B) can be matched with WfMC’s Interfaces 2-4 (Workflow Client Applications, Invoked Application and Workflow Interoperability). Both groups of interfaces (A and B) are specified in WSDL. End-user can interact with YAWL system via a web browser.
4.2.3 Other business process engines

Talking about business processes described in other languages, some literatures show the way to map to BPEL. C. Ouyang, M. Dumas, A. H.M., and W. M. P. van der Aalst [48] proposes a BPMN-to-BPEL translation method, which identify not only perfectly block-structured fragments in BPMN models, but quasi-structured fragments that can be turned into perfectly structured ones and flow-based acyclic fragments that can be mapped onto a combination of structured constructs and control links. Moreover, this article addresses the general issue arising when dealing with translation between graph-oriented and block-structured flow definition languages. W. M. P. van der Aalst and K. B. Lassen [49] provide a mapping from Workflow Nets (WF-nets) to BPEL. This mapping builds on the rich theory of Petri nets and can also be used to map other languages (e.g., UML, EPC, BPMN, etc.) into BPEL. S. Pornudomthap and W. Vatanawood [50] propose a YAWL-to-BPEL transforming procedure, which transforms the YAWL process into a meta-model using control flow graph. The procedure supports both the well-structured and non-well-structured process/workflow patterns. These methods still remain in the theoretical level, so there are still no practical transformations available yet. However, these theories provide a feasible way for the architecture described in Figure 30 to interact with other business processes besides BPEL and YAWL.
4.2.4 On-line advertising example

To demonstrate the value of the architecture (in Figure 30), we show how it can be applied to the on-line advertising example (in section 3.3). Figure 34 and Figure 35 show the customized business process monitoring architecture for customer A and customer B (see Table 16), respectively.

In the on-line advertising case, the monitoring variable is the current value of the number of clicks on the ad of the customer. The customer wants to monitor the MV along the campaign, therefore, the anchoring point enable and disable monitoring are the start and the end of the campaign, respectively. As shown in left most part of Figure 34, “GetClicks”, “Enable” and “Disable” are exposed via WSDL to the normalized message router, so that the customer can decide the anchoring points as well as the number of the clicks on the ads.

The customer A (see Figure 34) chooses “persist” as the way to manage data, so the BPEL engine exposes the “persist” via WSDL to the NMR, so that all the MV will be stored along the campaign. Talking about the way to supply data to the monitoring clients, there are three options that must be configured, they are “Read” (The MV will remain there after the customer gets data), “Pull” (The “Supply Data” will be triggered by the customer) and “Periodic” (The data will be supplied in a fixed time interval). These three services will also be exposed via WSDL to the monitoring clients.
The customer B (see Figure 35) chooses the same way to get data from the business process as the customer A. However, the way to supply data to the monitoring clients is different. It chooses “Consume” (The data will be cleared after the customer gets data), “Push” (The “Supply Data” will be triggered by the internal trigger) and “Threshold” (Only those which meet the requirements will be supplied to the monitoring clients). Like customer A, these three services will also be exposed via WSDL to the monitoring clients.

4.2.5 Analysis of the proposed architecture

The advantages of the proposed architecture are shown in Figure 30. Each of the components can work properly without knowing the definition of other separate components. The resolve of interconnection problem releases the burden of the BPEL designer and customization system designer, as a result the easiness for deployment of the platform will be significantly improved. Moreover, different business process language and programming language can be embedded into this architecture. It also provides the software designer to view the modules as centralized despite of the fact that they may be distributed in a complex computer network. However, nothing comes for free; the use of this architecture brings the complexity to build up the environment. The pros and cons are summarized in Table 18.

<table>
<thead>
<tr>
<th>Pros:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heterogeneity of implementation language</td>
</tr>
<tr>
<td>Resolve interconnection problems that are a result of different implementation languages used for components.</td>
</tr>
<tr>
<td>Heterogeneity of deployment platform</td>
</tr>
<tr>
<td>Resolve interconnection problems that are a result of different deployment platforms used for components.</td>
</tr>
<tr>
<td>Pros</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>Heterogeneity of technical interconnection</td>
</tr>
<tr>
<td>Distribution of the set of modules</td>
</tr>
</tbody>
</table>

**Cons:**

| Complexity | Take a lot of the burden of complexity from the shoulders of a software designer in a complex, distributed environment. |

*Table 18 - Pros and cons of the architecture*
4.3 Quality and Evaluation

4.3.1 Criteria
As defined in the IEEE Glossary of Software System Engineering Terminology [51], quality can be treated as the degree to which a system, a component, or a process meets client or user needs or expectations. Quality evaluation is challenging since measurement must consider different perspectives for different stakeholders. The factors that will affect quality are described as quality attributes. After studying the literature about the Bohem’s Quality Model [52], McCall’s Quality Model [53] and ISO 9126 Quality Model [54], the quality attributes framework is proposed as shown in Figure 36.

![Figure 36 - Evaluation framework](image)

The evaluation framework consists of seven phases, maintainability, usability, performance, availability, reliability, reusability and security. In the following seven paragraphs, we will elaborate more in detail about these criteria.

**Maintainability**: Maintainability consists of four main components: analyzability, changeability (corrections, improvements or adaptations of the software to changes in environment), stability (load or endurance testing) and testability (the degree to which a software artifact supports testing in a given test context).

**Usability**: Usability is a metric to measure how easily, comfortably and effectively an end user to perform and learn the tasks at hand. It involves both the architectural and non-architectural aspects. On the one hand, the non-architectural aspects include making user interface clear and easy to use; on the other hand, the architectural aspect includes for example, support for cancelation and undo commands.
**Performance**: Performance is a general metric to measure how a system performs in terms of execution time and resource utilization. According to Smith and Williams [55], performance refers to responsiveness: either the time required to respond to specific events or the number of events processed in a given interval of time.

**Availability**: Availability is a metric to measure the degree to which a system is in a specified operable and committable state. It is concerned with system failure (unavailability) and its associated consequences. It is defined as the relative amount of time the system functionality is accessible. System breakdowns and malfunctions have a negative effect on availability.

**Reliability**: Reliability is a metric to measure the ability of the system to maintain its functions in routine circumstance. It can be also defined as the mean amount of time that the software is available for end users as indicated by several sub-attributes, such as maturity, fault tolerance, and recoverability.

**Reusability**: Reusability is the likelihood a segment or a component that can be used again to add new functionalities with slight or even no modifications at all. It is concerned with the extent to which a program or parts of the program can be reused in other applications.

**Security**: Security is defined as the degree of protection against danger or damage. It can ensure the integrity and confidentiality of exchanged information or information of a server by prohibiting intruders accessing unauthorized information.

### 4.3.2 Analysis

#### 4.3.2.1 High Cohesion

Cohesion is a measure of how well a component fits together and how strong-related or focused the responsibilities of a single module are. It means the component should implement a single logical entity or function. There are various levels of cohesion, in the order from the lowest level to the highest levels are as follows: coincidental cohesion, logical association, temporal cohesion, communicational cohesion, sequential cohesion, functional cohesion and object cohesion. (Check Table 19)

<table>
<thead>
<tr>
<th>Type of Cohesion</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>coincidental</td>
<td>Parts of a component are simply bundled together.</td>
</tr>
<tr>
<td>logical association</td>
<td>Components which perform similar functions are grouped.</td>
</tr>
<tr>
<td>temporal</td>
<td>Components which are activated at the same time are grouped.</td>
</tr>
<tr>
<td>communicational</td>
<td>All the elements of a component operate on the same input or produce the same output.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sequential</th>
<th>The output for one part of a component is the input to another part.</th>
</tr>
</thead>
<tbody>
<tr>
<td>functional</td>
<td>Each part of a component is necessary for the execution of a single function.</td>
</tr>
<tr>
<td>object</td>
<td>Each operation provides functionality which allows object attributes to be modified or inspected.</td>
</tr>
</tbody>
</table>

Table 19 - Type of cohesion

The business process monitoring customization infrastructure employs the functional cohesion, since parts of the module are grouped because they all contribute to a single well-defined task of the module. Simply put, functional cohesion is when a function does only one thing, which can be thought to be the strongest and best type. For instance, in our project, the push or pull function is an example of functional cohesion. Each of these functions returns a supply-data type. The functions are named so that you know exactly what to expect. No surprises, no frills.

Cohesion is thought to be desirable as when a change has to be made, it is localized in a single cohesive component. More conceptually put, cohesion is a measure of abstraction that means developers do not need concern themselves with the internal working of module. Go back to the evaluation phase in section 4.3.1, it makes the system easy to maintain and improve the general performance. If a module is in high cohesion, it’s feasible to reuse that module in another system. The decreased development time through reuse improve the availability of the system.

4.3.2.2 Loose coupling
Coupling generally refers to the act of joining two things together, such as the links in a chain. There are two types of coupling, namely loose coupling and tight coupling. Shared variables or control information exchange usually leads to tight coupling, which is thought to be a bad design. However, if the components/modules are unlikely to affect each other, it’s a loose coupling. It can be achieved by state decentralization (as in object) and component communication via parameters or message passing.

The business process monitoring customization infrastructure is a loose coupling design. Each component communicates via WSDL, which is a type of message passing technology, with the Enterprise Service Bus (ESB). Each component can be implemented by different programming language if they can expose themselves via WSDL, and is independent from other components. Go back to the evaluation phase in section 4.3.1, it makes the system easy to maintain and reliable. Each component can be reused by another system, especially in an ESB environment. The decreased development time through reuse improve the availability of the system. Each component can focus on its own performance improvement; as a result, the performance of the system is significantly improved. Least but not last, each component can only communicate via WSDL, and the security is improved as well.
4.3.2.3  **Well defined Interfaces**
In the software engineering, there is a principle stating that the overall number of the communication channels between modules should be as small and few as possible. This principle means every module should communicate with as few others as possible.

The business process monitoring customization infrastructure has well defined interfaces. In the client side, the end user can only access the system via WSDL using the browser, which improves the usability of the system. Each component/module has an interface, via WSDL to communicate with the central ESB. This makes the world cannot see anything behind the interface, ensuring that subtle errors cannot be introduced when using connected modules, which as a result improves the security of the system.

4.3.2.4  **Short summary**
In general, a high cohesion, loose coupling and well defined interfaces improve the general quality of the system. As shown in Table 20, high cohesion enhances the system from the maintainability, performance, availability and reusability perspective. In the meanwhile, compared to high cohesion, the loose coupling design achieves extra benefits from the availability and security perspective. Least but not last, the well-defined interfaces provide a better user interface which improves the usability and enhance the system security.

<table>
<thead>
<tr>
<th></th>
<th>High cohesion</th>
<th>Loose Coupling</th>
<th>Well defined interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintainability</td>
<td>+</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Usability</td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Performance</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Reusability</td>
<td>+</td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>Security</td>
<td></td>
<td></td>
<td>+</td>
</tr>
</tbody>
</table>

*Table 20 - Summary of the evaluation*
5 Summary and conclusions

In this thesis, we present a conceptual framework for the modeling of mass-customized business process monitoring infrastructures. The framework identifies a set of orthogonal dimensions and options along which the customer monitoring requirements may vary. For each option, we provide patterns that model the process and data aspect of the monitoring infrastructure. The customized process monitoring infrastructure is then modeled through the combination of patterns from the dimensions.

Then, we are looking at a possible implementation of our framework. Three-tier architecture is proposed, the Enterprise Service Bus (ESB) acts as the middleware. On one hand, the client side is implemented in Java and can communicate with ESB via WSDL. On the other hand, this architecture well supports BPEL and YAWL via WSDL. In our thesis, it’s already proven that the implementations of all the patterns are practically feasible and the architecture is conceptually feasible.

Thus, the implementation of the architecture and further extension of the framework to a generic framework would be the focus of our future work.
6 Reference


Appendix

List of Figures

Figure 1 - Roadmap of the thesis ................................................................. 3
Figure 2 - Roadmap of the thesis ................................................................. 4
Figure 3 - Business process customization ................................................... 5
Figure 4 - PAIS lifecycle evolution ............................................................... 5
Figure 5 - Business process models (with variants) for complaints handling ..... 6
Figure 6 - Business process models (process injection) for complaints handling .. 7
Figure 7 - Three levels of customization ...................................................... 14
Figure 8 - High-level view of CQoS architecture ......................................... 15
Figure 9 - Structure of business process customizer .................................... 21
Figure 10 - Roadmap of the thesis .............................................................. 22
Figure 11 - (a) Monitoring scenario; (b) Business Process Monitoring Lifecycle ........................................ 25
Figure 12 - Interfaces between MI, MC and BPE .......................................... 27
Figure 13 - Pattern for anchoring point implementation ................................ 28
Figure 14 - Monitoring patterns for μ ≥? dimension options ................................... 28
Figure 15 - Monitoring patterns for the MI-MC <-> dimension options ............. 29
Figure 16 - Patterns for MI≥ dimension options ......................................... 30
Figure 17 - Advanced patterns for MI≥ dimension options .......................... 31
Figure 18 - Excerpt from the on-line advertising process ............................... 31
Figure 19 - Customized business process monitoring for Customer A ............ 33
Figure 20 - Customized business process monitoring for Customer B ............ 34
Figure 21 - Roadmap of the thesis .............................................................. 35
Figure 22 - Observer pattern ...................................................................... 36
Figure 23 - Class diagram .......................................................................... 37
Figure 24 - Log in ....................................................................................... 38
Figure 25 – Monitoring variables selection screen ........................................ 39
Figure 26 - Monitoring variables selection screen 2 .................................... 39
Figure 27 – Pattern selection ...................................................................... 39
Figure 28 – Result block ........................................................................... 40
Figure 29 - All patterns (left for the push, right for the pull) ......................... 41
Figure 30 - Architecture ........................................................................... 43
Figure 31 - How BPEL works ..................................................................... 45
Figure 32 - YAWL architecture [46] .............................................................. 46
Figure 33 - Reference model of the Workflow Management Coalition ............... 47
Figure 34 - Customized business process monitoring architecture for Customer A .. 49
Figure 35 - Customized business process monitoring architecture for Customer B .. 50
Figure 36 - Evaluation framework .................................................................. 52
Figure 37 - Roadmap of the thesis .............................................................. 56
List of Tables

Table 1 - List of I-options.................................................................................................................. 9
Table 2 - Sample list of complex I-options ......................................................................................... 9
Table 3 - List of PI-options.................................................................................................................. 10
Table 4 - Provop change operations .................................................................................................. 11
Table 5 - Provop options constraints ................................................................................................ 12
Table 6 - Source of variability.............................................................................................................. 12
Table 7 - Benefits of each framework................................................................................................. 13
Table 8 - Benefits of each framework ................................................................................................ 16
Table 9 - Constraints table .................................................................................................................. 18
Table 10 - Resources to be customized .............................................................................................. 18
Table 11 - Three capabilities for successful mass-customizer [23]....................................................... 19
Table 12 – Capabilities for successful monitorability mass-customizer ............................................... 23
Table 13 - Monitoring dimensions ..................................................................................................... 25
Table 14 - Monitoring options ............................................................................................................ 26
Table 15 - Possible monitoring dimension values for the IP address MV............................................ 32
Table 16 - Selected monitoring dimension options by Customers A and B ........................................... 33
Table 17 - Activities in BPEL .............................................................................................................. 45
Table 18 - Pros and cons of the architecture ......................................................................................... 51
Table 19 - Type of cohesion ................................................................................................................ 54
Table 20 - Summary of the evaluation ................................................................................................. 55