INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

A Domain Specific Language for Digital Libraries' Interoperability

João António Neves Edmundo

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente:	Prof. Alberto Silva
Orientador:	Prof. José Borbinha
Vogal:	Prof. André Vasconcelos

Outubro de 2011

Acknowledgments

I'd like to thank Eng. Gilberto Pedrosa for his time and important input in understanding the REPOX framework and the domain of digital libraries. Also, Eng. Hugo Manguinhas for its support on the development of the SVG library for GWT. My friends from the Taguspark Campus, for their infinite patience in helping me clear my thoughts and reviewing the texts, as well as cheering me up when the work seemed to never end; my parents, António Alberto Edmundo and Maria Isabel Edmundo, and my sister, Ana Isabel, for their continuous support and unshakable faith in my capabilities;

Abstract

Humans, in order to create, share and improve knowledge on business processes, need a common, readable and preferably visual notation. In addition, since Internet is more and more a platform for global application sharing, the requirements for Web applications concerning workflow execution, interaction, aesthetics and Web service integration are steadily increasing. In this paper we propose a Domain Specific Language, implemented as an extension of the standard BPMN 2.0 language specifically to the domain of digital libraries' interoperability, and use it to define and execute data harvest processes. To support it, we designed an architecture and implemented a real case of a computational environment based on the *jBPM* and GWT frameworks. This approach allowed us to provide a web-based, natural and flexible way not only to create and manage domain specific processes, but also to monitor each process execution, allowing process managers to understand the process history, its current state and possible future execution. In our opinion, creating specific languages for domains, and putting activity of interest in its visualized context, makes the user knowledge more comprehensive.

Keywords

Process Orchestration; BPMN 2.0; Specialization; Workflow Infrastructure; Domain Specific Language;

Resumo

Os seres humanos a fim de criar, partilhar e melhorar o seu conhecimento sobre processos de negócio, precisam de uma notação comum, legível e de preferência visual. Para além disso, e visto que a Internet é cada vez mais uma plataforma para partilha global de aplicações, os requisitos para aplicações Web relacionadas com *Workflow*, interação, estética e integração de serviços Web estão a aumentar progressivamente. Neste trabalho, propomos criar uma Linguagem de Domínio Específico, implementada como uma extensão da linguagem padrão BPMN2.0, mas especificamente para o domínio da interoperabilidade de bibliotecas digitais, e usá-la para definir e executar processos de recolha de dados. Para apoiá-la, desenhámos uma arquitetura e implementamos um caso real de um ambiente computacional com base em ferramentas como o *jBPM* e *GWT*. Esta abordagem permitiu-nos criar uma solução Web mais natural e flexível, não apenas para criar e gerir processos específicos do domínio, mas também para monitorizar cada execução do processo, permitindo aos gestores de processos perceber o histórico do processo, o seu estado atual e possível futura execução. Na nossa opinião, a criação de linguagens específicas para cada domínio, e colocando o interesse da actividade no seu contexto visualizado, faz com que o conhecimento do utilizador seja mais abrangente.

Palavras-chave

Orquestração de Processos; BPMN 2.0; Especialização; Infra-estruturas de Workflow; Linguagem específica de domínio;

Contents

List of Figures

List of Tables

1

Introduction

1.1. Motivation

Libraries, archives, museums and other cultural heritage organizations face the need to share their resource description information (usually data sets commonly named “metadata” in this domain), in international initiatives such as Europeana[1], TEL[2] and EuDML[3]. In these scenarios, organizations willing to share data but having a system not supporting natively the commonly required OAI-PMH protocol [18] [35] is an important constrain. Commercial and open-source solutions for this problem exist, but the first imply investments not always possible, and using open-source software might require some technical expertise for local customization, often not found in the staff of those organizations. Also, new emerging scenarios for transfer not only of data sets but also the contents of the resources referenced by these data sets (for example, the harvesting/ingest of the full-text of the documents described in the data sets) require the support for more sophisticated harvesting and aggregation processes.

1.1.1. Digital Libraries Domain Main Concepts

In this domain of digital libraries interoperability the concept of “data provider” means an entity that is willing to provide to a third party a collection of descriptive data sets (commonly named as “metadata”) describing information resources, and eventually also the transfer of the complete or partial referred information object (as thumbnails, full-text, etc.). The third party harvesting that data is commonly named of “service provider” if it is the perceived last entity in the value chain, or is named “aggregator” if it is an intermediary.

Other scenarios exist where roles are mixed. For example, TEL is in itself a service provider that harvests data from data providers consisting of national and university libraries, with the purpose of maintaining a resource discovery service in the TEL portal. However, TEL also plays the role of aggregator for Europeana, by providing to this more generic service the same data it harvests from its data providers. EuDML is also a service provider, which harvests data from open access digital libraries specialized on mathematics, and has the intention to, in the future, became also an aggregator for Europeana.

[1] Europeana –The European Digital Library -http://dev.europeana.eu/

[2] TEL - The European Library- http://www.theeuropeanlibrary.org

[3] EuDML– European Digital Mathematics Library- http://www.eudml.eu/

1.1.2. Data Aggregation in the Digital Libraries Domain

REPOX [10] is an open-source framework to address the problem of data aggregation in the digital libraries domain. It was designed and developed for convenient usage by both intended data providers and service providers, requiring little technical knowledge and effort, thus supporting a fast start process (installation and configuration). It is focused on common metadata interoperability scenarios, offering not only the publication and harvesting, but also support for metadata transformation. In this sense, REPOX also is a convenient tool for service providers.

However, the initial releases of REPOX only provided metadata harvesting services that are used within built-in processes. These processes cannot be shared between REPOX installations neither edited nor extended (to harvest objects' contents', for example) without programming knowledge. So, a new system's design was necessary, for which not also new frameworks to develop web applications and interfaces must be explored, but also find new ways to define and manage these harvesting processes. Therefore, the main objective of this work is to develop an easily deployable and portable graphical web interface for a data set aggregator to manage and monitor data harvesting processes, Also, because humans specialized in the domain have to define and monitor those processes, the solution also must propose a domain specific language (DSL) that represents concepts they already know, which will allow for a more productive and natural performance. Figure 1 shows a summary of the main concepts of this dissertation's context.

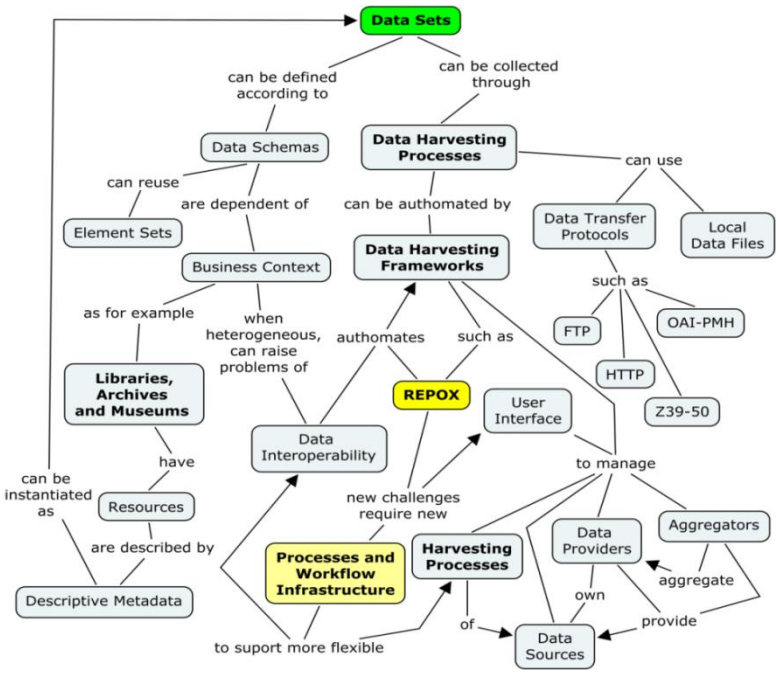


Figure 1. A concept map of the domain model.

1.2. Objectives

The main objectives of this dissertation are to:

- Create a Domain Specific Language (DSL) to support the processes of interoperability in digital libraries, extending the BPMN 2.0 notation to allow the visual process orchestration for domain experts.
- Define a Schema Definition Language (XSD) for process management and monitoring extending the BPMN 2.0 semantic XSD[4].
- Create a web Graphical User Interface (GUI) adequate to the process management and runtime visual monitoring of process instances defined by the proposed DSL.
- Create an extensible processing architecture, capable of orchestrating BPMN 2.0 processes to extend REPOX, namely, make it possible to replace its current hard-coded process architecture by a more flexible environment.

In sum, this research seeks to develop a solution to orchestrate business processes using a DSL derived from the BPMN 2.0 notation in scenarios of data aggregation and systems interoperability in digital libraries.

1.3. Main contributions

The main contributions of this dissertation are:

- A definition of the main concepts for interoperability in digital libraries and their visual representation through a DSL.
- A web framework for process orchestration customized to a DSL on interoperability in digital libraries, but flexible and extensible enough to allow the straightforward creation of new types of activities, concerning both their visual notation and actual behavior.

The results here proposed were validated in data aggregation scenarios in the projects SHAMAN[5], TEL, EuDML and Europeana.

[4]BPMN 2.0 Semantic definition - http://www.omg.org/spec/BPMN/2.0/20090502/Semantic.xsd

[5]SHAMAN - http://shaman-ip.eu/shaman/

1.4. Dissertation outline

Following the Introduction in Chapter 1, is the Related Work in the Chapter 2, where we start by investigating and comparing the most relevant frameworks using business process management and how and why they do it. Also, we investigate techniques used in the creation of DSLs and examples where it is used in extending the BPMN 1.0. Subsequently we examine the BPMN 2.0 and its more relevant characteristics to this dissertation. Finally we make an analysis of recent technologies used for the development of web applications. Next, in Chapter 3, we analyze the problem and retrieve a set of requirements which we use in our proposed DSL for digital libraries' interoperability described in Chapter 4. After that, in the Chapter 5, we present the components model of the solution, and detail how it was implemented. To complete the dissertation, we evaluate our work through real scenarios in Chapter 6, and finally, in Chapter 7, we present the conclusions that wraps up the solution to the problem initiated in this introduction.

2

Related Work

In this chapter we start by analyzing the relevance of business process management as an organization's management approach in order to understand its value for process orchestration. Also, we investigate the use of BPMN as a visual notation for defining business processes, and the guidelines used for defining domain specific languages based on BPMN. Next, we examine the currently used workflow technology based on BPMN and how can it be enhanced through UI design. Finally, we evaluate the recent technologies used in the development of web applications.

2.1. Business Process Management

Many people consider Business Process Management (BPM) to be the "next step" after the workflow wave of the nineties. BPM is a management approach focused on aligning all aspects of an organization with the wants and needs of clients. It is an approach that promotes business effectiveness and efficiency while striving for innovation, flexibility, and integration with technology. BPM attempts to improve processes continuously, being therefore described as a "process optimization process." It is argued that BPM enables organizations to be more efficient, more effective and more capable of change than a functionally focused, traditional hierarchical management approach [40].

2.1.1. BPM lifecycle

"Any mature discipline is usually organized in the form of a lifecycle, with phases that are logically separate from each other but have well defined hand-off points to move from one phase to the next. BPM too can be defined, at a high level, as a lifecycle consisting of well-defined phases"[6]. Figure 2 shows the various phases that might constitute a typical BPM lifecycle.

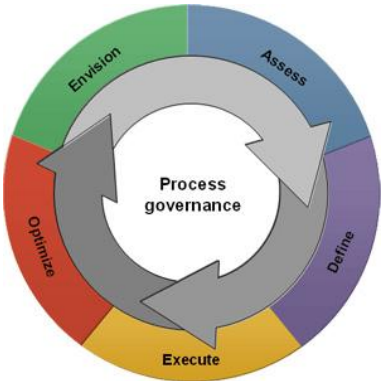


Figure 2. BPM lifecycle[6].

[6] From http://www.ibm.com/developerworks/library/ar-arprac6/

These phases are described as follows:

- **Envision** – Functions are designed around the strategic vision and goals of an organization.
- **Assess** – The "as-is," or current state of the enterprise as applicable to process design and development, is analyzed.
- **Define** – Where the "to-be," or future steady state, enterprise business processes are developed (design, implementation, deployment, and management). The proposed improvement could be in human-to-human, human-to-system, and system-to-system workflows, and might target regulatory, market, or competitive challenges faced by the business.
- **Execute** – The high-level definition of the business, and IT architecture and its components, are actually modeled, built, integrated, assembled, deployed, and monitored in their respective run times.
- **Optimize** – The various aspects of the enterprise architecture are monitored, managed, and optimized for better performance, and to meet the business and IT metrics used to define the success of the enterprise operations.

In conclusion, BPM provides a framework that enables enhanced control and management of core business processes across an organization. Also, an enterprise can integrate the business functions they've built over the decades by using BPM tools, techniques, technologies, best practices, and business processes as the fundamental construct. Consequently, the enterprise will be much more flexible, dynamic, and capable of integrating into the value chain of products, suppliers, and consumers. Therefore it can be in the middle of the chain as a value-addition node to the overall value delivery network.

2.1.2. BPM Notation and Business Process Execution Language

As a mean to visually represent BPM, the Business Process Modeling Notation (BPMN[7]) was created by the Business Process Management Initiative (BPMI[8]) and it is currently maintained by the OMG (Object Management Group) since the two organizations merged in 2005. This standard is widely used in the business process modeling industry [20]. BPMN defines a Business Process Diagram (BPD), which is based on a flowcharting technique very similar to activity diagrams from Unified Modeling Language (UML), tailored for creating

[7] BPMN – Business Process Modeling Notation - http://www.omg.org/spec/BPMN/2.0/PDF/

[8] BPMI – Business Process Management Initiative - http://www.bpmi.org/

graphical models of business process operations. A Business Process Model is subsequently a network of graphical objects, which are activities and the flow controls that define their order of performance, based on a flowcharting technique [41].

BPMN is also supported with appropriate graphical object properties that enable the generation of executable Business Process Execution Language (BPEL[9]). BPEL, short for *Web Services Business Process Execution Language* (WS-BPEL), is an OASIS [10] standard executable language for specifying actions within Business processes with Web Services. Processes in BPEL export and import information by using Web Service interfaces exclusively. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation [40].

2.1.3. BPMN 2.0

The current version of BPMN specification is 2.0[11]. It not only defines a standard on how to graphically represent a business process like BPMN 1.x, but also includes execution semantics for the elements defined, and an XML format on how to store process definitions. These last characteristics are a very important and innovative feature of BPMN 2.0 which grants this standard a prominent position in the industry [15] [30]. Also, BPMN 2.0 can be extended to include advanced features, and provides collaborative B2B (business-to-business) processes and private business processes. Furthermore, although a process can be defined through graphical tool (Figure 3), it is also possible to create a process using a XML file, according to the XML process format as defined in the XML Schema Definition in the BPMN 2.0 specification.

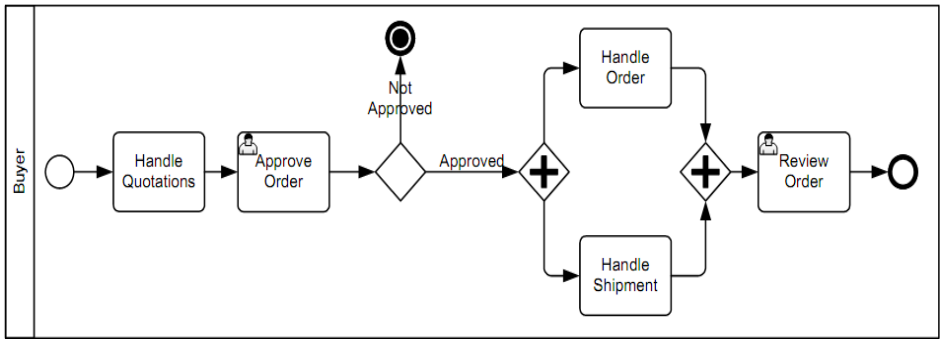


Figure 3. Example of procurement Process modeled in BPMN 2.0.[11]

[9] BPEL - http://en.wikipedia.org/wiki/Business_Process_Execution_Language

[10] OASIS - http://www.oasis-open.org/

[11] BPMN 2.0 - http://www.omg.org/spec/BPMN/2.0/PDF

2.2. Principles for the Definition of a DSL

Flexible and innovative business processes are one of the key elements that enable modern organizations to succeed. According to Janis Barzdins et al. [3], Mark Strembeck [34], and Marjan Mernik et al. [23], there is a growing need to consider new issues when implementing tools for domain specific languages with an orientation to the business process management. That's because nowadays business processes are not only modeled but also managed (meaning the process modeling tool has been integrated into some process management system which controls the process execution and integrates other parts of the information system). Consequently, in order to define, share and improve knowledge on business processes, humans need a standard way of describing them.

There are many business process modeling languages. Between the most popular we can find on one side the Unified Modeling Language (UML)[12] activity diagrams, which allowed Mehner [22], Artho et al. [2], and Xie et al. [43] to propose extensions to the UML to enable modeling of concurrent behavior, and on the other the BPMN, which allows process execution through its compiler to BPEL (Business Process Execution Language), which is executable by some BPEL engines.

However:

> "…most of these BPM languages or tools are often not very useful in everyday situations. Being very complex they are of course very useful for large enterprises. However, smaller and more specialized systems usually need only a small part of those facilities provided by the universal languages and tools. As a result, the usage of them tends to be too complicated" [3].

Therefore, specialized languages for narrow business domains are required, and that is where the concept of DSL comes into play. Although universal languages make advances towards specific tool builders (e.g., BPMN offers a possibility to add new attributes for tasks[13]), they can never give such wide spectrum of facilities as a DSL can. In addition, frequently there are already well accepted notations for manual design of processes in some business domains, and they can be adequately formalized by the DSL approach.

When building DSL tools in the field of BPM, some requirements must be taken in consideration. Generally speaking, a DSL tool consists of two parts – a domain specific

[12] UML – Unified Modeling Language - http://www.uml.org/

[13] Task definition in http://www.omg.org/spec/BPMN/2.0/PDF page 186

language it implements, and services it offer. For a BPM domain specific tool to be successful, it must be able to:

- Establish a connection to some external data source like a relational database.
- Convert a process definition in this DSL into specification for some process execution engine in the system.
- Generate some kind of reports from the model information.

Considering these issues is a crucial factor when designing a new DSL tool in the business process management context.

According to these principles some solutions were developed. Steen Brahe et al. [6] proposed the use of two tools (both Eclipse IDE plug-ins) to enable an enterprise to efficiently define and utilize their own Domain Specific Modeling (DSM) language. One tool, called *ADSpecializer* [6], can generate a UML profile and its tool support of a given application domain. The other tool, *ADModeler* [6], is used to create UML activity diagrams within such a domain-specific UML profile. According to the author, "General-purpose modeling languages are inadequate to model and visualize business processes precisely. An enterprise has its own vocabulary for modeling processes and its specific tasks may have attached data that define the tasks precisely." To make an example of the solution, the author starts by illustrating the power of defining a DSM language and a customized tool for a particular domain by looking at the processes in a human family. The *Family* language is defined starting by the determination of what specialized tasks are required to model the process, what are the attributes for these tasks, and what new data types do we need.

Task	Icon	Description
Transport		Transport family members to a destination using some kind of transportation, e.g. a car, a bus or a train
Clean		Clean a room. The cleaning can be of different types, e.g. vacuum cleaning, wash the floor etc.
Cook		Cook a meal. It must be specified which kind of meal should be created; breakfast, lunch or dinner
Shop		Do some specific shopping, such as groceries or clothes.
Relax		Take some time for watching TV, exercise or sleep. For the task it must also be specified for how long time relaxation can be done.
Nurse kid		Take care of the children, play with them, put them to bed, etc.

Figure 4. Custom tasks for the Family DSML. [6]

A similar definition paradigm is used when defining the semantics of some programming languages. That semantic approach, called “type system”, defines how a programming language classifies values and expressions into *types*, how it can manipulate those types and how they interact. The goal of a type system is to verify and usually enforce a certain level of correctness in programs written in that language by detecting certain incorrect operations. In most typed languages, the type system is used only to type check programs, but a number of languages, usually functional ones, infer types, relieving the programmer from the need to write type annotations. As a result of the need to define a DSL in the *Family* domain, the author created a set of tasks and data types to characterize it. Therefore, a set of new data types for the new language is created (Figure 5). Here the author defined only Enumeration data types, although it is possible to defined composite data types containing attributes of other data types.

Data type	Possible values
TransportationType	Car, Bicycle, Train, Bus
CleanType	Vacuum clean, Wash floor
RoomType	Kitchen, Toilet, Living room
MealType	Breakfast, Lunch, Dinner
ShoppingType	Grocery, Clothes, Lumberyard
ActivityType	Sleep, Play soccer, Watch TV
NurseType	Play, Bath, Change nappies, Put to bed

Task	Attributes	Type	Description
Transport	meansOfTransport destination	TransportationType String	Which transport? Where to go?
Clean	room cleanWhat	RoomType CleanType	What room to clean? What to clean?
Cook	Meal Persons	MealType Integer	Which meal to cook? Number of persons.
Shop	shopKind	ShopType	What to shop?
Relax	activity duration	ActivityType integer	What to do? How many minutes?
Nurse kid	activity duration	NurseType integer	What to do? How many minutes?

Figure 5. Data types (top table) and custom tasks (bottom table) for the *Family DSL*. [6]

After specified the required data types, it is expected to define the custom tasks and their attributes as shown in the previous figure. These tasks are then visually represented and described (Figure 4) for defining the specific *Family* language tasks.

The *Getting Home from Work* process was modeled in *ADModeler* using the Family language and can be found in Figure 6, which also illustrates the *ADModeler* working with the generated plug-in containing the Family language.

In the tool palette to the right in Figure 6, all the customized tasks shown are represented in a previously defined table (Figure 4) that describes the task name, icon and description for each task for the specific language. Also, a task instance can be dragged from the palette onto the model. Doing this made the tool more intuitive to use by a domain expert.

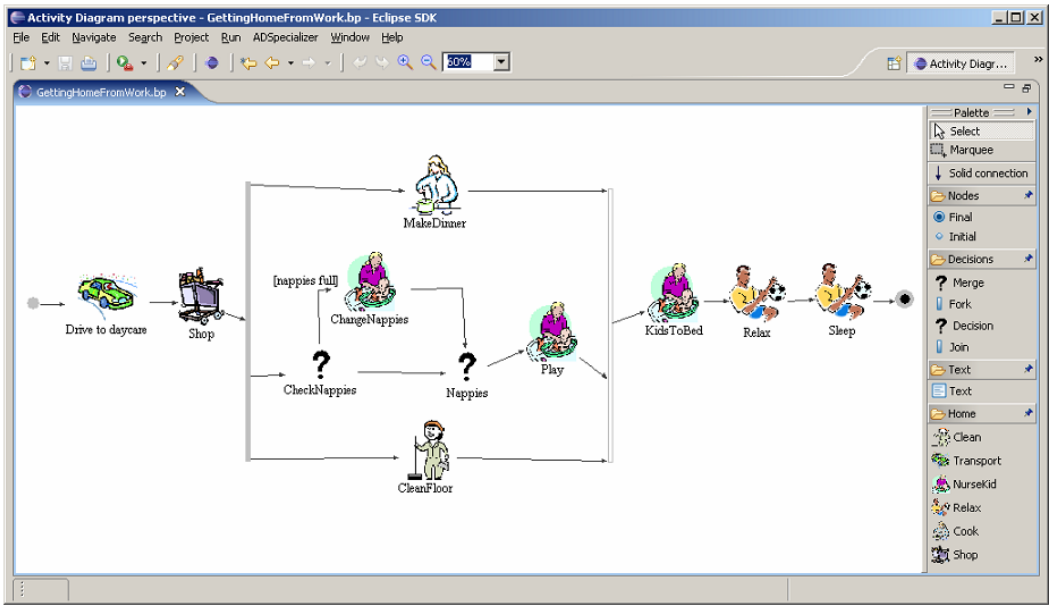


Figure 6. *ADModeler* with the Family DSM language extension and modeling the *Getting home from work* process. [6]

Focusing more the visual definition of a DSL (domain-specific visual language - DSVL), John Grundy et al. [13] developed *Marama*, a set of Eclipse plug-ins that create domain-specific visual modeling tools specified using high-level DSVL tool specifications produced from an existing meta-tool, *Pounamu* [44]. *Marama* allows users to rapidly specify or modify a desired visual language tool using *Pounamu* design tools and then have the tool realized as a high-quality Eclipse-based editing environment. Multiple users and multiple views are supported along with visual editing and complex behavioral specification support. *Marama* DSVL editors look and feel like other Eclipse graphical editors, use Eclipse code generation support, and can be integrated with and extended by other Eclipse plug-ins. Their specifications can, however, be modified on the fly using *Pounamu* allowing rapid trialing and deployment. Therefore, this approach allows the rapid implementation of a wide range of Eclipse-based domain specific language tools. Beyond all their benefits, the previous solutions are still Eclipse dependant and with limited extensibility when trying to create a web application.

An alternative solution for defining a DSVL was proposed by Dario Correal et al. [8]. It uses the concept of viewpoints to create additional visual elements depending on the role of the person defining the process. Therefore, it was created a DSL called *AspectViewpoint*, to generate viewpoint models using a vocabulary based on the workflow control patterns. Six of those proposed workflow control patterns were used: sequence, parallel split, synchronization, exclusive choice, arbitrary cycles, and merge. Finally, results show that by means of these

patterns, process designers found it easier to express, in a DSL, the requirements expressed by stakeholders. Particularly, process designers found useful the incorporation of the workflow patterns as part of the vocabulary of the language and the ability to define, in separate modules, the concerns expressed by the stakeholders. Also, process designers expressed that these characteristics facilitated to understand the effects of the changes introduced on the target process after the weaving of each viewpoint. Besides these several advantages, there is still the need develop a graphical interface for the *AspectViewpoint* language to define viewpoints directly into de process model diagram.

A DSL can also have a visual representation when it is applied, for example, to the business process modeling domain. Therefore, these visual DSL or Domain Specific Modeling Language (DSML) enters the domain of Visual Programming Languages. A Visual Programming Language (VPL) is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually. Also, a VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols used either as elements of syntax or secondary notation. For example, many VPLs (known as *dataflow* or *diagrammatic programming*) [5] are based on the idea of "boxes and arrows", where boxes or other screen objects are treated as entities, connected by arrows, lines or arcs which represent relations. VPLs may be further classified, according to the type and extent of visual expression used, into icon-based languages, form-based languages, and diagram languages.

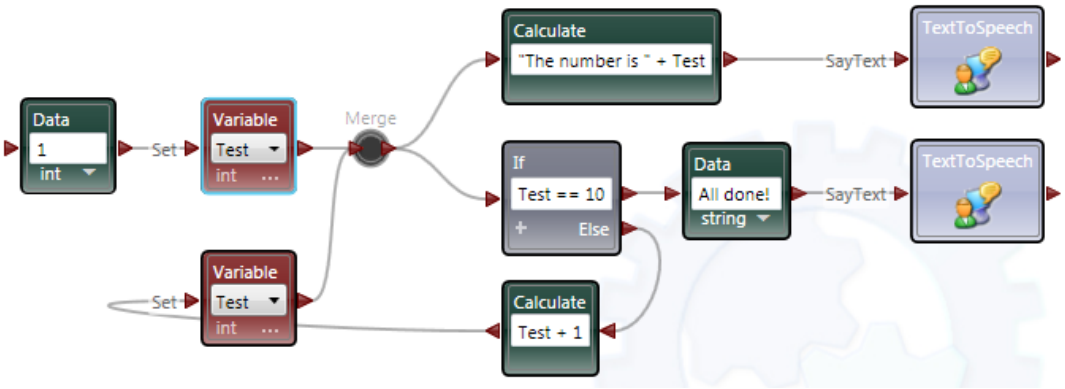


Figure 7. Microsoft's VPL dataflow.[14]

Visual programming environments provide graphical or iconic elements which can be manipulated by users in an interactive way according to some specific spatial grammar for program construction. Current developments try to integrate the visual programming approach

[14] From http://msdn.microsoft.com/en-us/library/bb483088.aspx

with dataflow programming languages to either have immediate access to the program state resulting in online debugging or automatic program generation and documentation (i.e. visual paradigm). This is exemplified by the Microsoft's VPL example in Figure 7.

In sum, these principles serve as guidelines proposals as to what a language designer should consider during the development of a DSL.

A different solution by Momotko [24] proposed extensions to the BPMN 1.0 that allow the monitoring of business processes. Therefore, the solution defines a process instance notation as an extension of BPMN. The underlying premise for this approach is the reuse of well-defined and commonly accepted concepts from the process definition level on the process instance level. In addition, in order to enable performers to check quickly what the current status of the process is, the current operational state of an activity instance is expressed as different color of the activity box (Figure 8).

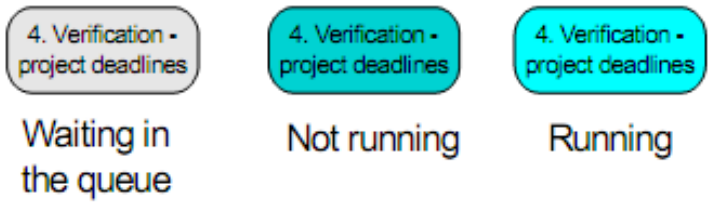


Figure 8. Representation of activity operational behavior. [24]

This solution made the author realize that introducing a simple but quite powerful subset of new elements to BPMN increased its functionality and didn't increase its complexity. As a result, it allowed a better performance on the monitoring of process instances (Figure 9).

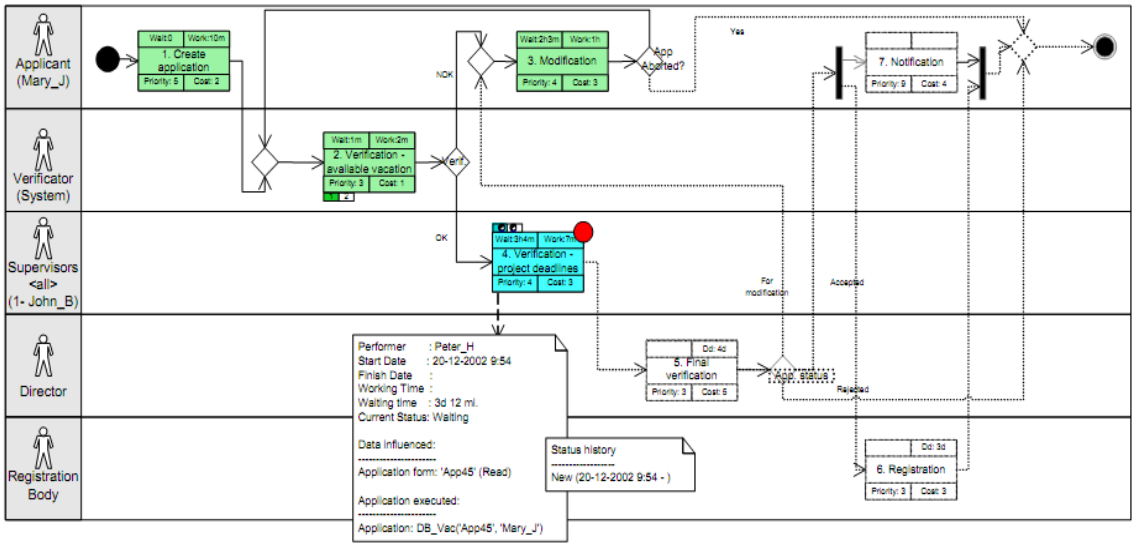


Figure 9. Example of a process instance execution using a BPMN extension. [24]

Furthermore, when extending a language in conformity with the BPM notation, it increases the opportunity to share process execution knowledge between the users of different BPM systems and thus between different organizations.

These approaches extend the standards in order to get improved performance by the domain experts (human users who are professionals in a particular domain).

2.3. Workflow technology used in BPM

Workflow technology is not only applied in traditional application areas of business process modeling and business process coordination, but also in emergent areas like component frameworks, inter-workflow, and business-to-business interaction [29]. Some companies perceived this and started developing technologies that allowed an easier business process creation, execution and management.

BizTalk Server[15] is Microsoft's Integration and connectivity server solution. BizTalk Server 2010 provides a solution that allows organizations to more easily connect disparate systems. It provides functions like: Enterprise Application Integration, Business Process Automation, Business-to-business Communication and Message Broker. BizTalk Server also provides connectivity between core systems both inside and outside the organization. Although it is very maturely developed software with a lot of functionalities, it is still a paid one, and only works on the Windows operating system.

IBM created the *WebSphere* [17], which not only supports managing the life cycle of business processes, navigation through the associated process model, and invokes the appropriate Web services, but also is compliant with BPEL. *WebSphere* allows the extension of BPEL to provide support of J2EE constructs; this enables developers familiar with the Java language to use Java features seamlessly within business processes, eliminating the gap between the Java world and the Web services world thus enhancing developer productivity. Although it is paid software, *WebSphere* allows for modeling not only automatic business processes but also manual ones, assigned to certain certified users.

In order to create an open-source approach of this kind of frameworks, the *JBoss*[16] community developed the *jBPM*[17], which is a workflow engine written in Java. In essence *jBPM* offers open-source business process execution and management, including an embeddable,

---

[15] BizTalk - http://www.microsoft.com/biztalk/en/us/default.aspx

[16] JBoss - http://www.jboss.org/

[17] jBPM - http://www.jboss.org/jbpm

light-weight process engine in Java, supporting native BPMN 2.0[18] execution and process modeling in Eclipse (Figure 10). Processes represent an execution flow, which is stored in a XML file. The graphical diagram of a process is used as the basis for the communication between non-technical users and developers.

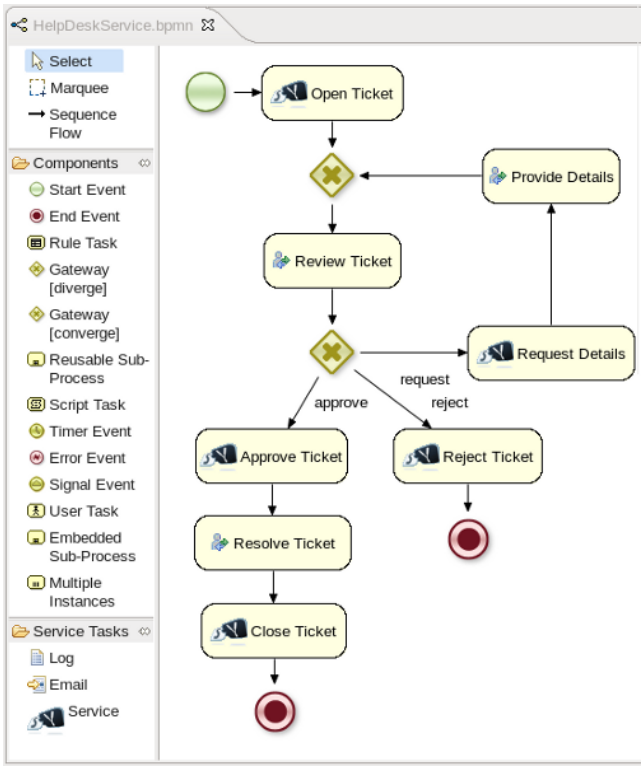


Figure 10. jBPM process definition in Eclipse.[19]

Some activities are automatic like sending an email, others can be wait states like human tasks or waiting for an external client to invoke a service method. *jBPM* manages and persists the state of the process executions during those wait state activities. *jBPM* is a widely used framework because of its flexibility and extensibility. *jBPM* favors a similar, layered approach as VPML (Visual Process Modeling Language): process models are designed graphically by domain experts (nodes, transitions). As a framework it can easily be extended with new node and transition types. When integrating *jBPM* into an application where only non-persistent workflows or business processes are required, the *jBPM* Java library is the only thing

[18] BPMN 2.0 - http://www.omg.org/spec/BPMN/2.0/

[19] From http://community.jboss.org/wiki/SwitchYardBPMComponent

required. For systems that require persistent workflows or business processes, the only additional requirement is a database supported by Hibernate[20].

One example of *jBPM* being integrated into a system is Alfresco[21], an open source Enterprise Content Management (ECM) solution. Alfresco uses *jBPM* to implement workflows for content. Also, it relies on its own interface to pull workflow information from *jBPM* and display it to the users in an intuitive way. Therefore, *jBPM* provides a good approach to metadata harvesting processes, because not only makes the current hard-coded approach to process representation more flexible through XML, but also is free, open source, easily integrated, and supports immediate and scheduled tasks, which will be used depending on the number of metadata records.

An integrated solution of process management is done by the *jBPM* console, which integrates the *jBPM* and an UI for process management (Figure 11). It allows the management of both process definitions and process instances.

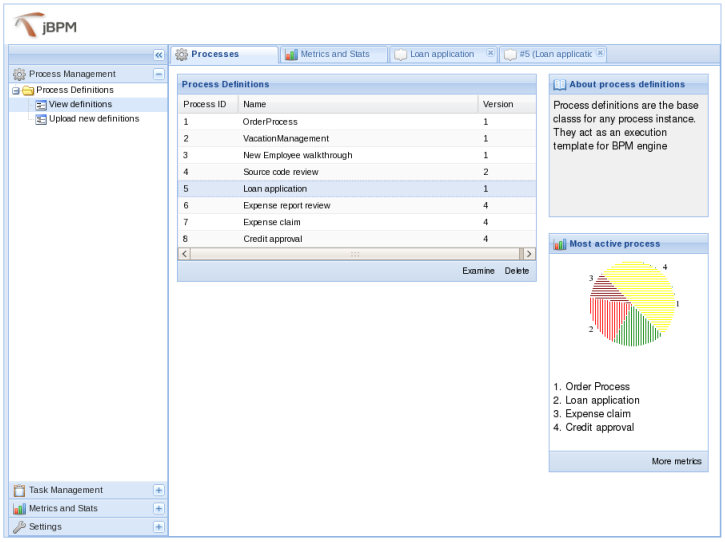


Figure 11. Example of a console UI using jBPM.[22]

Other open-source frameworks developed are the YAWL [38], Triana [36], Taverna [27], and the *Activiti*[23] BPM Platforms. This last solution was developed by some of the people involved in *jBPM* 1 to 4, and a lot like *jBPM*, provides a light-weight workflow and Business

[20] Hibernate - http://www.hibernate.org/

[21] Alfresco - http://www.alfresco.com/

[22] From http://edgarsilva.com.br/2009/01/08/preview-jbpm4/

[23] Activiti - http://www.activiti.org/

Process Management (BPM) Platform based on BPMN 2.0, and targeted at business people, developers and system administrators. It is still a very young project but promises to be competitive rival for *jBPM*.

To achieve a more natural and flexible way in the creation of business processes, and based on the BPMN, some approaches allow the definition of a business process through graphical association between activities, connected through flow controls. The *Signavio*[24] commercial project provides easy access to professional business process management recurring to web technology. It allows a graphical definition of business processes on a SaaS[25] approach, which also can incorporate employees or even business partners, suppliers and customers into the procedure of process design. A non-commercial open-source project, which served as basis for *Signavio*, is *The Oryx Project*[26], which is also a web-based editor for modeling business processes in various modeling languages like BPMN. *Oryx* is based on open internet standards, has a large developer community and permits the extension of new functions via a plug-in mechanism.

In sum, there are several solutions that use BPM as a management approach, and allow process definition through BPMN and execution with BPEL. Although some are commercial like IBM's *WebSphere* and Microsoft's *BizzTalk Server*, open source solutions start to appear. These solutions like *jBPM* and *Activiti* use BPMN 2.0 as the core process definition and execution language to their process engine, which gives them the flexibility to define complex processes. Although these frameworks give the user a process definition interface through Oryx, they still lack the process execution monitoring interfaces, which is an important issue for process managers [26].

2.4. Integrating Business Process Modeling and UI design

BPMN has over the last years appeared as a major approach for modeling process-oriented solutions (as described in the previous sections). In addition, the approach is meant to work well both towards human understanding and execution. Executability is normally based on a mapping of BPMN-models to BPEL and defining a form for each flow where the user is the source or target. According to Hallvard Trætteberg et al. [37] and Stefan Betermieux et al. [4]

[24] Signavio - http://www.signavio.com/

[25] SaaS – Software as a Service - http://www.wikinvest.com/concept/Software_as_a_Service

[26] The Oryx Project - http://bpt.hpi.uni-potsdam.de/Oryx

this often gives sub-optimal and inflexible user interfaces. He proposes a solution using BPMN for process and task modeling and Diamodl[27] for model-based user interface dialog design.

Another similar solution based on Diamodl developed by Renata Dividino et al. [9] shows how to maintain the consistency and integrity of the several correlated models in process service modeling, and demonstrates the integration of user interface design and business process models.

As shown in Figure 12, the UI needs to be aware of the state of the business process in order to set its behavior. This includes switching from one activity to another depending on the state. More importantly, the business process is bound exactly to the events triggered by UI with the intention of avoiding non-deterministic behavior. In other words, the business process must ensure that an event occurring during its execution is properly handled by the corresponding UI elements in the case of a human involvement is needed. Therefore, it was considered an event-based coordination to achieve the synchronization between two models.

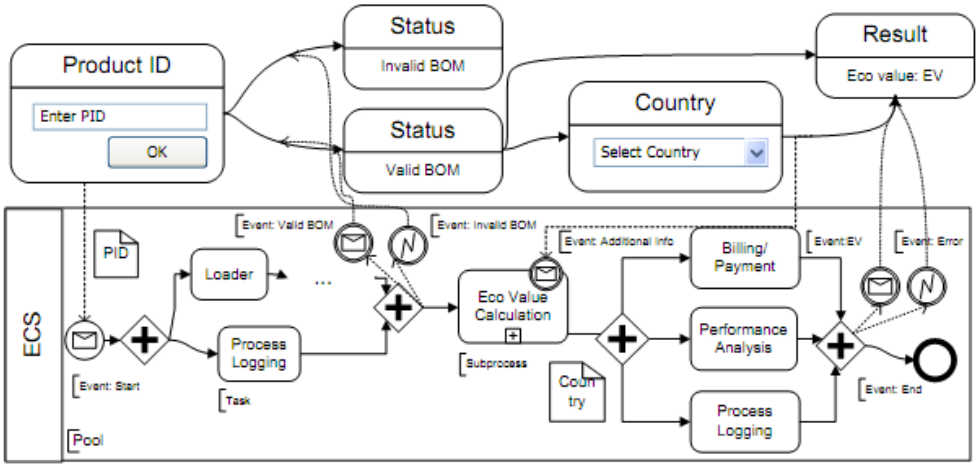


Figure 12. BPMN model showing the core elements and interactions with UI components. [9]

In sum, the proposed solutions try to create a seamless integration between process modeling and UI design, giving the user a more natural way of defining processes and input when modeling a BPMN process.

[27] Diamodl- http://www.idi.ntnu.no/~hal/research/diamodl

2.5. Technology for Web-Based Application Development

In order to develop a web application capable of creating a process orchestration framework, the right technology must be found. Nowadays the concept of Web 2.0 is becoming popular. Web 2.0 refers to the concept of new web applications that are interactive in nature and are intended to help people to collaborate and offer services for them, not just static HTML. This became possible, in part, by means of the Asynchronous JavaScript and XML[28] (AJAX) technology. In reality, AJAX represents several technologies, each flourishing in its own right, coming together in powerful new ways [11]. Ajax incorporates: standards-based presentation using XHTML[29] and CSS[30]; dynamic display and interaction using the Document Object Model[31] (DOM); data interchange and manipulation using XML and XSLT[32]; asynchronous data retrieval using *XMLHttpRequest*; and JavaScript binding everything together. In comparison with those old web applications constrained by HTML, Ajax based applications bring user interface and functionality with rich, responsive, intuitive, interactive and dynamic features entirely inside the browser without plug-in or other software required [28]. Besides rendering HTML and executing script blocks, the browser plays a more active role in processing HTTP requests and responses in these applications. Instead of traditional "click, wait, and refresh" user interaction, these rich internet applications show better performance and web experience since they could add or retrieve users' requests asynchronously without reloading web pages [14]. Also, knowing that presently Web sites and Web applications tend to rely quite heavily on client-side JavaScript to provide rich interactivity, particularly through the advent of asynchronous HTTP requests that do not require page refreshes to return data or responses from a server-side script or database system [19], this research will also focus on finding the one tool (or framework) that helps to develop JavaScript-based graphical interfaces.

The frameworks compared in this work were: Prototype+*script.aculo.us*[33] [12], jQuery[34], ExtJS[35], MooTools[36], Dojo[37] [33], Google Web Toolkit (GWT)[38] [21] [16], and ZK[39] [7]. In the next Table 1 the comparison results are shown.

[28] AJAX - http://www.adaptivepath.com/ideas/essays/archives/000385.php

[29] XHTML - http://www.w3.org/TR/xhtml1/

[30] CSS - Cascading Style Sheets - http://www.w3.org/Style/CSS/

[31] DOM - Document Object Model - http://www.w3.org/DOM/

[32] XSLT - http://www.w3.org/TR/xslt20/

[33] Prototype - http://www.prototypejs.org/

[34] jQuery - http://jquery.com/

Table 1. Comparison between web development frameworks.

	Prototype + *script.aculo.us*	**Jquery**	**ExtJs**	**MooTools**	**Dojo**	**GWT**	**ZK**
Developer Community	Medium	High	High	High	Low	High	Medium
Multi-Language Support	High	Low	Low	High	High	High	High
XML Parsing	High With Plugin	High With Plugin	High	Low	High	High	Low
Cross-Browser Support	High	High	High	High	High	High	Medium
Ease of Widget Extension	High	High	High	High	High	High	Low
Widget Collection	Medium	Medium with Plugin	High	Low	High	High with Plugins	High
Performance (overall)[40]	Low	Medium	Medium	Medium	High	High	Low
Overall modeling of complex UI interaction[41]	Low	High	Medium	Medium	Low	High	Medium
Minimal learning curve	Medium	Low	Medium	High	High	Medium	Low
Ease of Use (API)	Medium	Medium	Medium	Medium	Low	Medium	High
Documentation	Low	High	Medium	Medium	Low	High	High

Analyzing Table 1 it can be perceived that the Prototype + *script.aculo.us* and MooTools frameworks are the less satisfactory to our project. Although they have a medium learning curve and ease of use, their performance and overall modeling of complex UIs is way below the other frameworks'. Compared to those, the Jquery and ExtJs frameworks are suitable to build

---

[35] ExtJS - http://www.sencha.com/products/js/

[36] MooTools - http://mootools.net/

[37] Dojo Toolkithttp://www.dojotoolkit.org/

[38] GWT – Google Web Toolkit - http://code.google.com/webtoolkit/

[39] ZK - http://www.zkoss.org/

[40] http://blog.creonfx.com/javascript/mootools-vs-jquery-vs-prototype-vs-yui-vs-dojo-comparison-revised

[41] http://www.slideshare.net/hsplmkting/top-javascript-frameworks-comparison

complex UIs, have better documentation and learning curve, but still lack acceptable performance and multi-language support, which is very important since an international interface is needed so it can be used in several countries. Furthermore, the Dojo framework provides higher performance and a very big widget collection, but it is not suitable to build complex UI interactions, has a high learning curve and little documentation. Finally, this leaves us with GWT and ZK as our final framework solutions.

In ZK, it is easily perceivable that it requires less line coding, and allows a simpler data access compared to GWT client and server side programming. Also, it uses a markup language to program and design most of the interface, allowing for less programming knowledge and therefore more focus on interface design. Being ZK server centric makes all the UI processing and data retrieval to be done by the server. But if the server gets accessed by many users, which means a lot of server requests, it is more efficient to move the UI handling load out of the server into the user's laptop. Therefore, it is unnecessary to use server accesses just to update the look of the UI. This makes ZK a low performance framework compared to client sided ones like GWT. Also, even though it has a very high widget collection, their extension is very hard.

With GWT, though it has a medium learning curve and ease of use, it provides a high performance and extensible framework to build complex UI interactions. This permitted the growth of a large developing community that created many libraries to provide additional widgets and functionalities. Some examples of these libraries are the *Ext GWT*[42] and *Smart GWT*[43], which is a Java library for building rich internet applications with the GWT, leveraging existing enterprise skills with the GWT compiler.

[42] ExtGWT - http://www.sencha.com/products/extgwt/

[43] SmartGWT - http://code.google.com/p/smartgwt/

2.6. Summary

In this section, we learned the importance of BPM in an organization's efficiency, effectiveness and flexibility to adapt to changes. As a mean to visually represent BPM, the Business Process Modeling Notation (BPMN) was created which is based on a flowcharting technique very similar to activity diagrams from UML, tailored for creating graphical models of business process operations. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation [40]. The current version of BPMN specification is 2.0[11]. It not only defines a standard on how to graphically represent a business process like BPMN 1.x, but also includes execution semantics for the elements defined, and an XML format on how to store process definitions.

After this, we researched the importance in creating Domain Specific Languages. It showed us that although general languages can cover a large variety of cases, they add unnecessary complexity to specialized systems [3]. Therefore, specific languages for narrow business domains are required. As a result, some solutions presented by Steen Brahe et al. [6] and Momotko [24] show a set of guidelines used when defining a DSL, and present techniques for DSL creation based on BPMN, through the use of colors and custom icons.

To find a suitable approach to define and execute our DSL, we studied some workflow technologies. Although several commercial products like Microsoft's *BizzTalk* and IBM's *WebSphere* are widely used in the BPM domain, some open-source solutions also start to appear. These solutions like *jBPM* and *Activiti* use BPMN 2.0 as the core process definition and execution language to their process engine, which gives them the flexibility to define complex processes. However, they still lack the process execution monitoring interfaces and modeling input interfaces, which are important issues for process managers [26][4].

Finally, to create our web framework for process orchestration we searched for the right tool for the job. After comparing some web development frameworks like Prototype+*script.aculo.us* [12], jQuery, ExtJS, MooTools, Dojo [33], Google Web Toolkit (GWT) [21] [16], and ZK, we concluded that GWT, though it has a medium learning curve and ease of use, it provides a high performance and extensible framework to build complex UI interactions, eventually being the more fitting to our problem.

In the following chapter we will analyze the problem and through it define a set of goals and requirements to our DSL and framework.

3

Analysis of the Problem

In this chapter we fully analyze the problem from the meaning of metadata to the use of data exchange protocols in the digital libraries' aggregation and interoperability domain. As a result, we establish a set of goals and requirements currently used in data aggregation tools, which will be used in the definition of our DSL in the next chapter.

3.1. Metadata Representation

The emergence of the World-Wide Web has made available a multitude of autonomous data sources which can, as a whole, satisfy most of users information needs. However, it remains a tedious and long task for users to find the data sources that are relevant to their problem, to interact with each of those sources in order to extract the useful pieces of information which then eventually have to be combined for building the expected answer to the initial request [31]. A concept developed to support these tasks was to promote, for these purposes, the creation of *data about the data*, or usually called metadata.

In the digital libraries' domain, the metadata concept is usually referred to as a description of information resources available in libraries, archives, museums. These descriptions can be, on a basic level, the title, creation date, author, etc. of a certain historic document[44].

3.2. Digital Libraries Aggregation and Interoperability

The growing concern in the search for unique ways to represent metadata is driven by the final goal to achieve metadata interoperability. Metadata interoperability has a very broad meaning and entails a variety of problems to be resolved: on a lower technical level, machines must be able to communicate with each other in order to access and exchange metadata. On a higher technical level, one machine must be able to process the metadata information objects received from another. And on a very high, semantic level, we must ensure that machines and humans correctly interpret the intended meanings of metadata. During the last decades a variety of interoperability techniques have been proposed, among them being the introducing of new standardized metadata schemas and element sets.

3.2.1. Problem Statement

In the real example of the Europeana initiative[45], libraries, archives and museums throughout Europe are trying to share their information of catalogued resources using metadata. There are two major use cases that motivate this need:

[44] Descriptive metadata of a document using the Dublin Core Element Set - http://dublincore.org/documents/2001/04/12/usageguide/generic.shtml

- **Preservation**: The need to periodically transfer digital content from a data repository to one or more trusted digital repositories charged with storing and preserving safety copies of the content. The trusted digital repositories need a mechanism to automatically synchronize with the originating data repository.
- **Discovery**: The need to use content itself in the creation of services. Examples include search engines that make full-text from multiple data repositories searchable, and citation indexing systems that extract references from the full-text content.

Both the preservation and discovery use cases have been discussed in the context of Digital Library and Institutional Repository projects in The Netherlands, the UK and Germany[46]. The preservation use case is also emerging in the Archive Export/Harvest effort of the National Digital Information Infrastructure and Preservation Program[47]. The discovery use case has also emerged in the realm of web search engines, where both the sophistication of search technology and content coverage are competitive factors. This has led to growing interest by search engine providers in "deep web" content stored in digital libraries and institutional repositories, as exemplified by collaborations between OAIster[48], OCLC[49], arXiv[50], NSDL[51] and major web search engines.

The problem is that each institution uses different standards for their metadata representation, which makes it hard to create interoperability between them, in order to accomplish an easy exchange of data for a common use. In this same case, examples of those disparate metadata schemas are, for example the Encoded Archival Description (EAD) for archives [1], the Machine-Readable Cataloging standards (MARC/MARC21[52] and UNIMARC[53]) used for the representation and communication of bibliographic and related information in machine-readable form, and also the LIDO (Lightweight Information Describing Objects)[54] and

---

[45]Europeana - http://www.europeana.org/

[46] DINI - http://www.dini.de/

[47] Digital Preservation Initiative - http://www.digitalpreservation.gov/

[48] OAIster - http://www.oclc.org/oaister/

[49] OCLC - http://www.oclc.org/

[50] arXiv - http://arxiv.org/

[51] NSDL - http://www.nsdl.org/

[52]http://www.loc.gov/marc/

[53]http://www.ifla.org/unimarc

[54] http://www.lido-schema.org/schema/v1.0/lido-v1.0.xsd

the ESE (Europeana Semantic Elements)[55]. In this scenario it emerged the Dublin Core (DC) [39] metadata element set that provides adequate data for web resource discovery. As an example of the interoperability problems in digital libraries, we have an original metadata description of the document *Os Lusiadas*[56] which is digitally preserved using a UNIMARC schema[57]. However, to be able to share it in the TEL initiative, the metadata has to be changed to the OAI_DC[58] schema, which implies a transformation of the original metadata schema. Finally, a well-defined structure of the document and its images must be created to allow proper digital access. This is done using a METS file[59], which describes the entire structure of the document.

As shown in the previous example, based on the described and other related emerging standards new approaches to metadata interoperability became possible. However, an important related issue will be always the aggregation of those metadata sets from their original sources, which in some cases can have serious scalability requirements[60]. A fundamental piece to address that issue has been the OAI-PMH, precisely making use of the Dublin Core Metadata element set in its OAI_DC schema. This protocol provides an application-independent interoperability framework, in which *Data Providers* publish their data and *Service Providers* harvest it for end-application (such as for example for new added-value information retrieval services).

3.2.2. Harvesting protocols - OAI-PMH and Z39.50

OAI-PMH is based on client-server architecture, in which *harvesters* request information on updated records from repositories. A *harvester* is a client application that issues OAI-PMH requests, and is operated as a means of collecting (*harvesting*) metadata from repositories. A repository is a network-accessible server that can process the six OAI-PMH requests (Figure 13), and is managed by a *Data Provider* to expose metadata to *harvesters*. *Data providers* handle the deposit and publishing of resources in a repository, making the associated metadata available for harvesting by *Service Providers*. To allow various repository configurations, the OAI-PMH distinguishes between three distinct entities related to the

[55] http://www.europeana.eu/schemas/ese/ESE-V3.2.xsd

[56] Screenshot of the document *Os Lusiadas* - http://purl.pt/1/1/

[57] *Os Lusiadas* metadata in UNIMARC - http://urn.porbase.org/purl/unimarc/txt?id=1&agente=urn.porbase.org

[58] OAI-PMH Schema definition - http://www.openarchives.org/OAI/2.0/oai_dc.xsd

[59] Os Lusiadas structure description using a METS file - http://purl.pt/1/1/mets.xml

[60]The catalogues of most national libraries have records in the order of the millions.

metadata made accessible by the OAI-PMH: a *resource* (what the metadata is about); an *item* (a constituent of a repository from which metadata about a resource can be disseminated); and a *record* (metadata expressed in a single format). OAI-PMH specifies that *unique identifiers* are provided for items. Overall, using OAI-PMH, any client can have incremental harvest to any information stored in any data providers registered in the OAI, using a simple process (Figure 13).

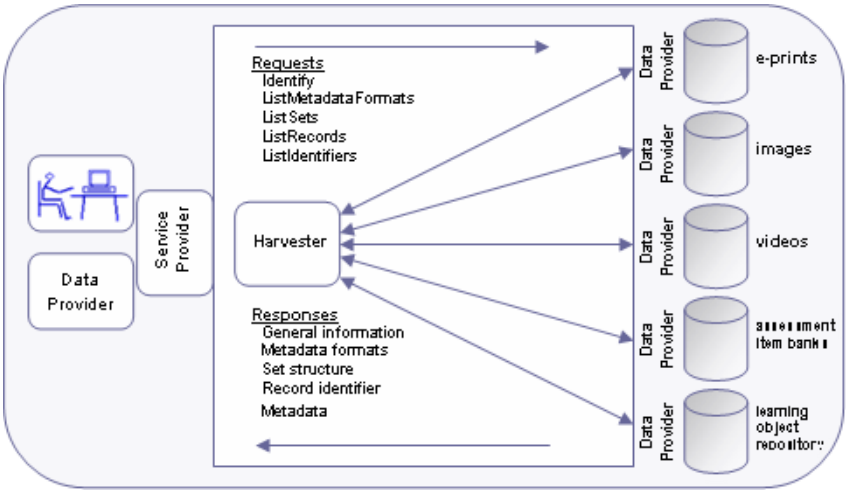


Figure 13. OAI-PMH overview functionality.[61]

Another protocol that is widely used within the digital library community for data harvesting and sharing purposes is the Z39.50. This protocol is a client–server protocol for searching and retrieving information from remote computer databases. It is covered by ANSI[62]/NISO[63] standard Z39.50, and ISO standard 23950. The standard's maintenance agency is the Library of Congress [64]. Z39.50 is widely used in library environments and is often incorporated into integrated library systems and personal bibliographic reference software. Inter-library catalogue searches for inter-library document loans are often implemented with Z39.50 queries.

3.2.3. Metadata Harvesting Frameworks

Based on the protocols described on the previous section, several projects like REPOX and the National Science Digital Library (NSDL) [42], started to develop new frameworks to automatically manage metadata harvesting. These harvesters collect the metadata of digital

[61] From http://wiki.cetis.ac.uk/What_is_the_OAI_Protocol_for_Metadata_Harvesting

[62] American National Standards Institute - http://www.ansi.org/

[63] National Information Standards Organization - http://www.niso.org/

[64] Library of Congress - http://www.loc.gov

libraries and use it to create a unified and transparent resource discovery system. Figure 1 shows a summary of the main concepts of this dissertation's context, where it is described that Data Sets (like for example the data collections of the EuDML[65] and several European libraries that contributed to The Europeana Project[66]) can be collected through data harvesting processes, which themselves can be automated by Data Harvesting Frameworks like REPOX. These frameworks are starting to deal with new challenges that require new user interfaces to manage Aggregators, Data Providers, Data Sources and Harvesting Processes. These harvesting processes are currently hard-coded within REPOX which leads to limited extensibility when simple changes are required. For example, a harvesting process is composed by the harvesting of the data from the data set, and then the publication of that data in a database for storage purposes. If we were to simply add a transformation task (or any other tasks) before publishing the data, this would require a change in the code itself, and therefore programming experience. Also, the portability of hard-coded processes is also hard to execute between REPOX instances. As a result, these processes need more flexibility to be able to add new tasks to a process, easily change input parameters for each task, and enable portability of these processes. This can be obtained through a new process and workflow architecture.

Faced with the previously described harvesting problems, some projects regarding digital library interoperability and data aggregation defined a set of goals and requirements.

3.3. Data Aggregation System Goals

In order to comprehend this domain, we also developed a graphical web interface for the current version of REPOX 2.0 (More detailed information can be seen on Appendix G). Therefore, through the gathered knowledge from the interface development and also the analysis of the concepts in the Europeana Libraries project [32], we were able to define the goals of a data aggregation system:

1. The management of the relevant information about the data providers and their OAI-PMH servers, data sets and descriptive metadata schemas.
2. The monitoring of the quality of service of the OAI-PMH servers.
3. The scheduling and management of automatic harvests.
4. The management of a central metadata repository of all harvested data sets.

[65] EuDML - European Digital Mathematics Library - http://www.eudml.eu/

[66] Europeana Project - http://www.europeana.eu/

5. The means of harvesting and indexing significant quantities of digitized material, including text, images, moving images and sound clips.

These goals define a service to allow data aggregation and interoperability between digital libraries.

3.4. Requirements for a DSL for digital libraries interoperability

Requirements specify the properties a system needs to fulfill according to its objectives and scopes. Therefore, requirements must result from the defined goals of the system and of the related analysis. As a result, the following requirements for the DSL where defined according to the goals described in the previous chapter:

[Req1]. Harvest a Data Set: It must be able to harvest any data set in any schema suitable to be represented in XML

[Req2]. Manage Data Providers: It must be possible to Create/Update/Delete Data Providers

[Req3]. Manage Data Sources: The system must be able to Create/Update/Delete Data Sources based on one of OAI-PMH, Z39.50, HTTP and FTP protocols

[Req4]. Manage Data Records: It must be possible to Save/Delete a Data Record

[Req5]. Data Transformations: It must be possible to apply translations to Data Sets to other desired schemas, with use of XSLT transformations. This process must result in a new Data Set

[Req6]. Data Transformations: Must be provided a mechanism to associate the automatic application of transformations, so Data can be published, and thus be harvested by Service Providers in schemas different of the schemas it was harvested from the Data Providers.

[Req7]. Harvest Content: It must be possible to support the harvest of the content of the information objects described and referenced by previously harvested data set records, through HTTP or FTP (this is mainly relevant in digital libraries interoperability to support central indexing services)

[Req8]. It must be possible to save the harvested data set records to a storage system (database or file system)

Through the previously described goals and requirements we were able to define a domain specific language for digital libraries' interoperability, which will be described in the next chapter.

4

Design of the Proposed DSL

In this chapter, and supported by the goals and requirements assembled on the previous chapter, we start by defining a set of main concepts in the digital libraries' interoperability domain, followed by the description and visual representation of the most relevant concepts and tasks proposed by our DSL.

4.1. Digital Library Interoperability main concepts

After analyzing the domain and its main goals and requirements (Chapter 3), we were able to create a set of concepts that define digital library interoperability as a whole:

- **Harvest**[67] - Based on standard schemas described in Section 3.2.1, many libraries and museums tried to establish a way to integrate their information, to be easily accessed and shared by clients, and so the concept of harvesting was born. Its meaning is the automatic gathering of data or metadata that is already associated with a resource, and which has been produced via automatic or manual means. Metadata harvested may be attached to a document (e.g., it may be encoded in the header of a Web resource), or it may be found in a metadata registry or database.
- **Store** – The act of store data is to persist a Data Provider/Data Source/Data Record so it can be referenced at a later time. This can be done by storing the data in a XML file or a database.
- **Publish** – After harvesting the data, make it available to any service provider who wants to access the harvested data. This access can be granted by making the data available through the OAI-PMH protocol, or by exporting it to the file system.
- **Transform** – Interoperability scenarios involving two or more digital libraries might require the transformation of metadata sets encoded in different schemas. Transform refers to the transformation of a Record set from one schema to another through a XSLT. This transformation can be done automatically or by a user. A specialization of a transformation might also represent the application of a filter (filter the data through a record's attribute value) or even a data analysis (statistic gathering) of a Record set.

These concepts define a digital libraries' interoperability system which gives the means for libraries and other cultural institutions to share their resources, regardless of their data formats. Additionally, we came to realize that these concepts are much alike the ones used on the

[67] An alternative name for this concept, sometimes referred in this document and used in some scenarios, is Ingest.

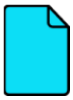
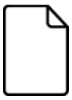
database and data warehousing domain known as Extract, Transform and Load (ETL[68]). They refer to the extraction of data from outside sources, transform it to fit operational needs, and finally load it to the final target (database or data warehouse). These concepts are very similar to our Harvest, Transform and Publish concepts.

Following the DSL definition principles (Section 2.2), we try to answer the questions: what specialized tasks do we require to model processes in the domain of digital libraries' interoperability, what are the attributes for these tasks, and what new concepts do we need to represent information exchange between them.

4.2. Domain Concepts Glossary

To characterize each concept within the DSL, a set of standard visual representations were used, and will be explained in this section.

Table 2. Information Entities in the DSL.

Name	Icon	Description
One Data Record		A Data Record (also commonly called in the context of digital libraries Metadata Record) is a structure of information describing one information resource
One Record Set		A Record Set is a collection of Data Records
One Record Subset		A Record Subset is in itself a Record Set made as a subset of a Data Record Set
One Data Provider		A Data Provider is an entity that publishes one or more Data Sources: the publication of a Data Source corresponds to make available a Record Set
One Data Source		The way by which a Data Provider makes available a Record Set, which can happen by multiple means (a file available from an FTP or HTTP server, an OAI-PMH service, etc.)

These entities, which represent a piece of data or a group of pieces of data with a unique semantic definition (Table 2), allow the exchange of information between the DSL tasks described in the next section. Next, in Table 3, some operations/actions are represented using some standard visual symbols.

[68] ETL – Extract, Transform, Load - http://www.webopedia.com/TERM/E/ETL.html

Table 3. Description of Operations/Actions in the DSL.

Name	Icon	Description
Create		Creation/Save/Edit a certain entity
Delete		Delete/Remove/Cancel an entity
Harvest/Get		The input of data into the system
Export		The output of data from the system
Schedule		The scheduling of a certain task
Transform		A transformation of a Data Record or a Data Set (usually, a conversion to a new format, according to a new schema)
Package		Grouping of several instances into one

Finally, the *Technology* concepts (names of the protocols applied in data harvest) used in this DSL are visually represented by their own written name since they don't have a standard symbol that represents them: **OAI**, **Z39.50** and **Folder**.

These previously described concepts lead to the visual representation of our tasks in the DSL, presented in the next section.

4.3. Domain Specific Tasks

After describing the main concepts of the domain, defining the information entities, and choosing the visual representation of the concepts in our DSL, a set of tasks are proposed. We start by presenting the tasks related to Data Providers, in Table 4.The set of tasks regarding the Data Sources is proposed in Table 5. Finally Table 6 shows the tasks for the Data Records.

Table 4. DSL proposition tasks and their characteristics for Data Providers.

Task	Icon	Description
Create/Edit Data Provider		Creates/Edits a Data Provider in the system through its name, country and description, returning a Data Provider entity
Delete Data Provider		Deletes an existing Data Provider (considering its identifier)

Table 5. DSL proposition tasks and their characteristics for Data Sources.

Task	Icon	Description
Create/Edit Data Source - OAI		Creates/Edits an OAI-PMH data source using an OAI-PMH repository urn and data set, and returns a Data Source entity.
Create/Edit Data Source - Z39.50		Creates/Edits a Z39.50 data source. It can be of type *Timestamp*, *Id List*, or *Id Sequence* according to the Z39.50 Type field (Appendix A.b).
Create/Edit Data Source - Folder		Creates/Edits a Folder data source. It can be of type FTP, HTTP or Folder as a pathname in an actual file system, according to the FType field (Appendix A.b).
Delete Data Source		Deletes an existing Data Source with a given identifier
Harvest Data Source		Initiates a harvesting session of a Data Source through its identifier, and return a Record Set entity with the harvested data
Harvest Data Source Sample		Initiates a harvesting session of a Data Source, ingesting only a pre-defined number of records.
Delete/Cancel Task		Deletes a scheduled or already running export or harvest task of a Data Source through its identifier
Export Data Source		Exports the records from a given Data Source using its identifier, and a user-defined number of records per file
Create/Edit Schedule Harvest		Creates/edits a scheduled Data Source harvest task starting on a given date and with a certain period
Create/Edit Schedule Export		Creates/edits a scheduled Data Source export task starting on a given date and with a certain period

Table 6. DSL proposition tasks and their characteristics for Data Records.

Task	Icon	Description
Save Data Record		Saves one new record using its identifier, content and the Data Source identifier it should belong to, and return the new Data Record entity
Delete Data Record		Deletes one existing Data Record using its identifier
Publish Record Set		Publishes a Record Set to a database or file system
Get Record Set		Retrieves a Record Set from a Data Source with a given identifier, and returns the corresponding Record Set entity
Transform Record Set		Transforms a Record Set with a given schema to another schema. A specialization of a transformation might also represent the application of a filter or even a data analysis of a Record set
Package Data Records		Groups one or multiple Data Record into one Record Set entity
Harvest Data Record Full-Text		Harvests the full-text content of one given Data Record with a given identifier
Harvest Record Set Full-Text		Harvests the full-text content of a given Record Set using its associated Data Source identifier

A more detailed description about the previously described tasks, including their parameters and return type, can be seen in Appendix A.

Altogether, the information entities and the previously described tasks that were defined based on the main concepts on Digital Library Interoperability, allow the definition of the DSL.

Finally, in order to define and execute processes using this DSL, we developed an architecture based on BPMN 2.0, which will be described in the chapter 5.

5

Implementation of the Solution

In this chapter we describe the proposed solution to create an extensible architecture that enables the orchestration of business processes based on the DSL described in the previous chapter. Additionally, we explain how the development process was managed and why some of the choices were made.

5.1. Process Orchestration Architecture

5.1.1. jBPM as a process workflow engine

As described in Section 2.3, JBPM is an open source WfMS (Workflow Management System) suited by JBoss. Initially, jBPM processes were created using a proprietary language of process definition called jPDL (jBPM Process Definition Language). jBPM5 is the latest version of the jBPM project. This version is based on the BPMN 2.0 specification and supports the entire life cycle of the business processes (modeling, executing, monitoring, and management). State management with jBPM is based on a graph with nodes and transitions that make up the definition of a process. jBPM5 suite provides embeddable, light Java process engine, supporting native BPMN 2.0 execution.

Being jBPM only a business process engine, it doesn't support threading in parallel cases, and it simply runs BPMN 2.0 defined processes. Therefore it has no concurrency controls for the process instance data. For instance, process variables cannot be locked when accessed nor are they implicitly locked by the engine. Truly parallel executions would lead to race conditions due to that. This is a common design trade-off when it comes to BPM engines. It is possible to avoid all concurrency control pitfalls (deadlocks, race conditions, starvation, consistency problems, etc.) with a single thread of execution for each process instance. Business processes are supposed to be long-running but are also supposed to be waiting for some event to happen most of the time and should not be compute intensive by themselves. So, CPU should never be a bottleneck when executing a single process instance.

Another problem we faced was the visual definition of the business processes. As described in Section 2.3, jBPM deals with this issue using an eclipse plug-in, which compels users who want to visually define a business process to acquire different software to create a process. This not only breaks the seamless interaction with the process management UI but also isn't a web solution, which limits portability.

The only component JBoss provides is a web interface for jBPM (Figure 14) that only supports process definition management; it doesn't support visual process definition, custom input UI during process definition, nor detailed process monitoring.

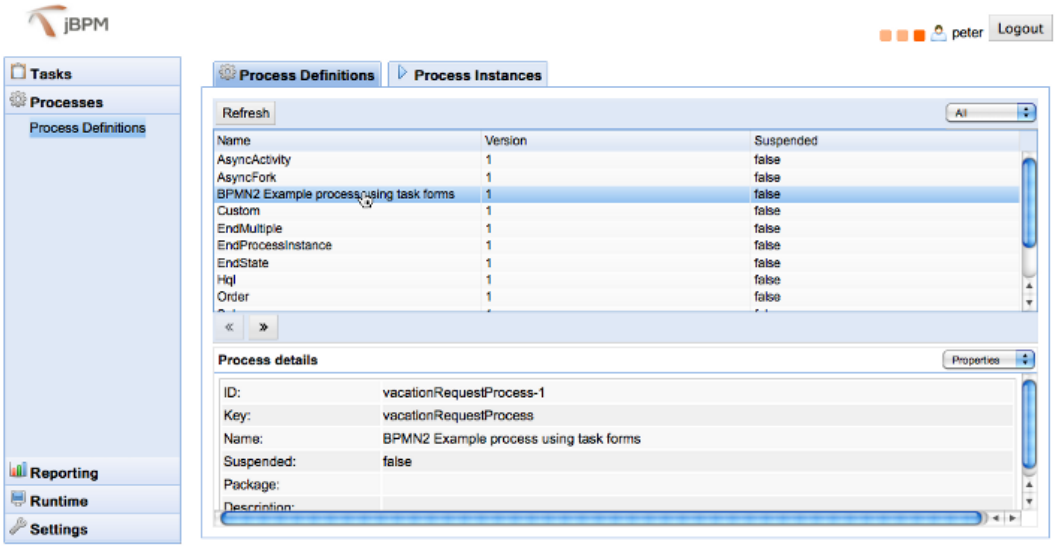


Figure 14. jBPM process definition management.[69]

These issues led us to research for possible web frameworks for BPMN 2.0 process definition.

In the domain of open-source visual business process definition, *The Oryx Project* and *Signavio* are solid frameworks. (For more detailed information please see Section 2.3) Though they are well defined BPMN 2.0 web editors, their extension to a specific domain language can be hard, especially when one wants to integrate the editor result with a process engine; create customized input variables and input methods; and allow process runtime monitoring. This led us to create our own web designer for process definition.

5.1.2. The Developed Architecture

Due to the limitations described in the previous sections we were compelled to develop a new architecture that would enable us to:

- Define business processes using a custom UI;
- Manage business processes and process instances;
- Runtime monitor of process instances;
- Execute parallel processes and parallel tasks within a process;
- Easily define the services of a process task;
- Easily create new process tasks.

[69] From http://www.diybl.com/course/3_program/java/javaxl/20100719/452489.html

Although the developed prototype uses the tasks defined on this thesis' proposed DSL (Section 4.3), its architecture can be easily extended to contain new languages and services due to its service oriented approach (Figure 15).

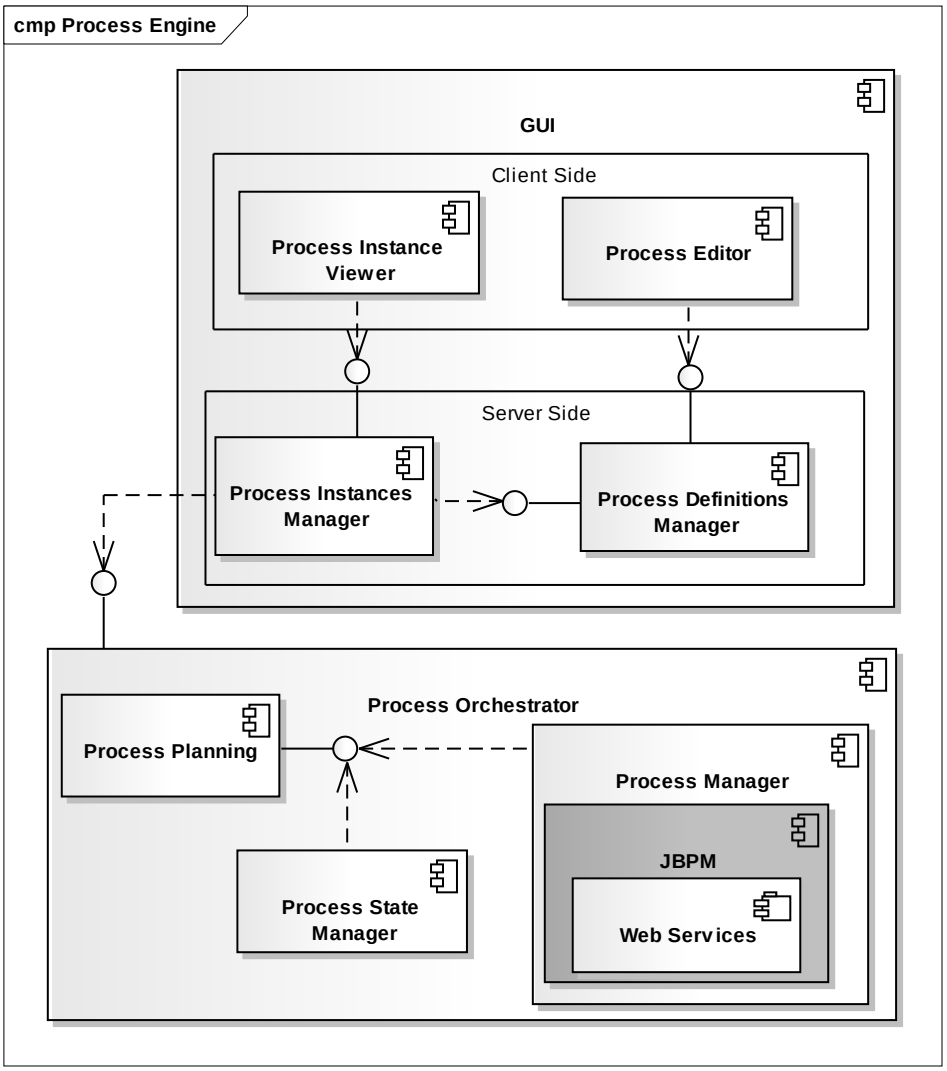


Figure 15. Process Engine architecture for process orchestration.

The core of the Process Engine is the Process Orchestrator, composed by a Process Manager that launches new processes in the JBPM engine and the Process State Manager which monitors the state of each process. Both these managers share the access to a common list of processes, managed by the Process Planning.

Each process consists in an orchestration of a set of Web Services that are registered within the JBPM. The Process Orchestrator is constantly monitoring all the running processes.

The Process Orchestrator provides an interface so that new processes can be added. These processes can be defined visually using the DSL based on the BPMN 2.0 notation, which is then coded in XML, according to the format defined by the OMG – Open Management Group.

The web application is composed by a client side containing a Process Editor that supports the visual definition of new processes, and by a Process Instance Viewer that allows the runtime monitoring of each process instance.

The processes defined on the client side are persisted in an extended BPMN 2.0, through the Process Definitions Manager. Initialized process instances are started through the Process Orchestrator, called by the Process Instances Manager.

This architecture allowed us to create an extensible web process framework that enables process orchestration from its definition to execution and runtime monitoring.

5.2. BPMN 2.0 Extension

In this section we describe how we extended the BPMN 2.0 and what base components were reused to achieve an integrated solution.

5.2.1. BPMN 2.0 Base Components

Being this DSL an extension of BPMN 2.0, we started by choosing which BPMN 2.0 main components we should use to help define our processes. As a result, and according to the analysis on usage of BPMN on different areas reported in [25], it was concluded that BPMN is used in groups of several, well-defined clusters, but less than 20% of its vocabulary is used regularly. The author also suggests that only a small subset of BPMN components has emerged, and is described in Table 7.

Table 7 represents the most commonly used BPMN components in process definition. Therefore we chose to include them in our own DSL based processes to provide a seamless adaptation for users familiar with BPMN process definition, and for new users, an already proven natural technique of defining processes [25].

5.2.2. BPMN 2.0 semantic extension

In order to create a DSL that would be executable in the jBPM engine we developed a BPMN 2.0 extended language based on the BPMN 2.0 semantic definition[70].

[70]BPMN 2.0 Semantic definition - http://www.omg.org/spec/BPMN/2.0/20090502/Semantic.xsd

Table 7. Most commonly used components in BPMN.

Name	Visual Representation	Description
Sequence Flow		The Sequence Flow connects two shapes in the process flow and defines the execution order of activities. It indicates the path to take from one shape to another. The sequence flow may also have a symbol at its start (a small diamond) that indicates one of a number of conditional flows from an activity while a diagonal slash indicates the default flow from a decision or activity with conditional flows. We also extended the sequence flow to a **dashed arrow**, in order to represent **data flow** (information exchange) between tasks.
Task		Tasks are a subtype of an Activity, which is a process step that can be atomic (Tasks) or decomposable (Sub-Processes) and is executed by either a system (automated) or humans (manual). A Task represents a single unit of work that is not or cannot be broken down to a further level of business process detail without diagramming the steps in a procedure. The visual notation of a Task is represented by a rounded-corner rectangle.
Start Event		A Start Event is a subtype of an Event, which is represented with a circle and denotes something that happens (rather than Activities which are something that is done). Icons within the circle denote the type of event (e.g. envelope for message, clock for time). Events are also classified as Catching (as in, they might catch an incoming message to Start the process) or Throwing (as in, they might throw a message at the End of the process). The Start Event acts as a trigger for the process; indicated by a single narrow border; and can only be *Catch*, so is shown with an open (outline) icon.
End Event		An End Event is a subtype of an Event described in the previous table entry. It represents the result of a process; indicated by a single thick or bold border; and can only *Throw*, so is shown with a solid icon.
Gateway		A Gateway is represented by the familiar diamond shape and is used to control the divergence and convergence of Sequence Flow. Thus, it will determine traditional decisions, as well as the forking, merging, and joining of paths. Internal Markers will indicate the type of behavior control. Gateways are split into the Basic (blank) Gateway, and an extended Gateway set, which comprises Data and Event based XOR, Inclusive-OR, **Exclusive**, and **Parallel Gateways**.

5.2.2.1. Process Definitions

The start and end events described in the previous section are semantically represented in BPMN 2.0 by a simple `<startEvent>` and `<endEvent>` element correspondently. Both have two attributes: “name” – represents some notes that can be added to the node, including its user-defined name; “id” – represents the unique identifier of the node.

The sequence flow component is represented by a `<sequenceFlow>` element that also has the “name” and “id” attributes and in addition has the “sourceRef” attribute that represents the id of the source element the sequence flow is connected to, and finally the attribute “targetRef” which represents the id of the target element the sequence flow is connected to.

As described in the previous section there are several gateways, but in our solution we only used the Parallel Gateway to allow the parallel execution of tasks, and the Exclusive Gateway to support decision making during the process. Its semantic representation is composed by a root element `<parallelGateway>` that, as the previous components, has the “name” and “id” attributes, but also the “gatewayDirection” attribute which can be: “Diverging” – more sequence flow connections going out then coming in; “Converging” – more sequence flow connections coming in the going out.

jBPM provides a mechanism that allows users to specify not only a task’s input and output variables, but also what that task is going to do by associating a work item handler (*IngestWI* in Figure 16). When a process reaches a certain task, that task’s work item handler is called and it executes its pre-defined behavior with the given input, producing a possible output.

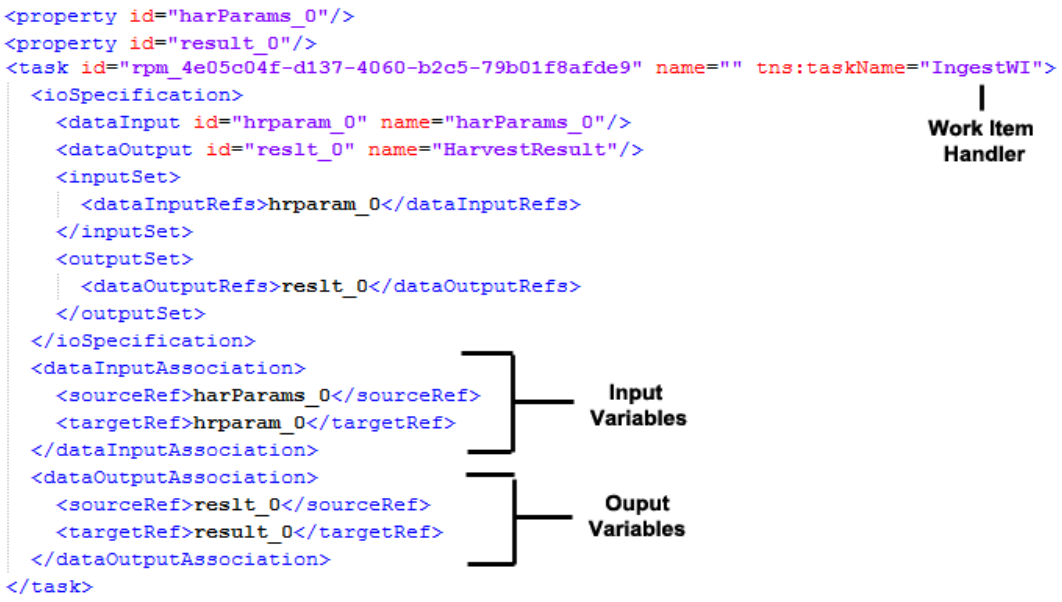


Figure 16. Definition of a semantic harvest task and associated input and output variables.

When initiating a process in jBPM engine, the chosen input must be passed in the format of a Map<String, Object>. As a result, to pass a value to the task defined in Figure 16, we can pass any Object value with a key value of “harParams_0” so that it will be processed within the work item handler. The input and output variables of a process are defined with the tag `<property>` at the beginning of the process and are used to exchange values between tasks within the process. This input can be given by the user before starting the process or can come from a previously executed task in the process workflow. Finally, when a work item handler concludes its execution, it can return a result in the same format as the input. (Ex: In the case of Figure 16, the work item handler will complete the task and write the return value with the key “HarvestResult”, this will lead to a saving of this result in the process variable “result_0” which can be now used by another task in the workflow. This kind of variable mapping shown in Figure 16 between a task’s input and output variables and its process variables/properties is needed to assure the execution of the process within the jBPM engine.

So far we described how to use the basic workflow components and define each task to represent the behavior we want extending the BPMN 2.0. However this data only allows us to execute the process within the jBPM, lacking the full information to preserve a process. As a result, we decided to continue extending the BPMN 2.0 to contain additional information (Figure 17).

```
<processGUI processRef="exec_process_ref" id="process_gui_id">
  <startEventShape eventRef="exec_process_start_shape_ref" x="6" y="8"
      name="" id="start_event_gui_id" state="NOT_PASSED"/>
  <activityShape activityRef="exec_process_activity_ref" type="INGEST"
      x="14" y="8" name="" id="activity_shape_gui_id" state="NOT_PASSED"/>
  <endEventShape eventRef="exec_process_end_shape_ref" x="23" y="8"
      id="end_event_gui_id" state="NOT_PASSED"/>
  <sequenceFlowConnector sequenceFlowRef="exec_process_sequence_flow_ref"
      targetRef="exec_process_target_ref" sourceRef="exec_process_src_ref"
      sourceGateIndex="2" targetGateIndex="0" name=""
      id="sequence_flow_gui_id" state="NOT_PASSED"/>
</processGUI>
<parameters>
  <parameter type="INGEST" taskID="exec_process_activity_ref" paramID="harParams_0"
      resultID="result_0" sendMail="true" dsID="arquivocmlx" fullIngest="true"/>
</parameters>
```

Figure 17. Visual and input information of a process.

As shown in Figure 17, we added two new elements to the process definition: `<processGUI>` – contains information about the process visual design; `<parameters>` - contains the input parameters given by the user while defining the process.

In the <processGUI> element represented on the previous figure we can see four types of shapes. These shapes represent each component described in section 5.2.1. All of them have the following attributes:

- <shapenameRef> - References to their corresponding execution process node;
- <x> and <y> - The coordinates X and Y of the component in the process definition grid;
- <id> - An unique identifier;
- <state> - A state that represents the current state of the node. This attribute can have the following values:
    - NOT_PASSED - if the node has not been reached yet;
    - PASSED - if the node has been successfully executed;
    - WORKING - if the node is currently executing;
    - FAILED - if the node failed its execution;
- <name> - Allows the adding of user defined notes to the node.

Some elements have specific attributes. In this example the <activityShape> represents a task component described in section 5.2.1 but, to distinguish different task types it has the <type> attribute, which in this case defines the task of type “INGEST”. This attribute can have different values depending on the different tasks represented in the system.

The other added element was the <parameters> which can have any number of <parameter> elements depending in the number of tasks used in the process. In this last element there are some required attributes:

- <type> - corresponding task type;
- <taskID> - corresponding task id in the process execution definition;
- <paramID> - input variable id used by this task in the process execution definition;
- <resultID> - output variable id used by this task in the process execution definition;
- <sendMail> - either if a mail should be sent to the user with the result of this task after its completion.

Adding to these main attributes each task may have their own. As shown in Figure 17, an ingest task has the <dsID> which represents the identifier of the data source that is going to be ingested, and the <fullIngest> that allows to perform a new ingest of all the records or update the

existing ones. Finally, other attributes can be added depending on the input required for each task. Every input of each task is given by the user through a custom UI described in the next section.

Through this extension based on BPMN 2.0, we were able to create a preservation format for process definitions that included the following information about the process: execution, visual design and parameters (A complete process definition of a simple harvest process can be seen in Appendix B).

5.2.2.2. Process Instances

After defining a process, we can create several instances of execution using that definition. These instances are represented in our solution as new XML files, and contain a copy of the `<processGUI>` and `<parameters>` elements described in the process definition (Figure 18). As a result, different instances of the same process definition can be created and have their own state of execution.

```
<processInstance xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
    xmlns:param="http://www.omg.org/spec/BPMN/20100524/MODEL" state="READY">
  <processGUI id="process_rpm_dfe2aa70-58f9-4913-9220-7180f4940fe7_gui"
    processRef="process_rpm_dfe2aa70-58f9-4913-9220-7180f4940fe7">
      ...
  </processGUI>
  <parameters>
      ...
  </parameters>
</processInstance>
```

Figure 18. Process Instance definition.

As shown in Figure 18, each process instance has a reference do its process definition (`<processRef>`), and also a `<state>` attribute that shows the current state of the process execution. This attribute can have the following values:

- READY - the process is ready to be started;
- RUNNING - the instance if currently running;
- FAILED - at some point the execution has failed;
- COMPLETED - the execution was successfully completed.

As described in the previous session the `<processGUI>` element contains all the visual design information of the elements of the process, including their current state. This state is updated in the process instance according to the process execution results (An example of a process instance can be seen in Appendix C).

Associated with each process instance file we created another XML file that saves the log of the process. This log can contain any messages related to the process including: starting time for each workflow component; duration of each task; task input or output values. These messages can be sent by any task implemented in the system at any time, allowing them to send enriched feedback on the results of the process execution to the user (An example of a process instance log file can be seen in Appendix 0).

All the previously described process instance attributes are important to send visual feedback to the user, and thus enabling a more effective and natural performance on process monitoring.

Finally, the `<parameters>` attribute in the process instance definition is used to pass the given user input to the jBPM engine when a new process is started. As described in section 5.2.2.1, if some tasks in the process have any input variables their value must be assigned when a new process execution is started in the jBPM engine. Therefore, we use the parameters defined in the process instance to fill the input variables and then execute the process definition through referenced by the `<processRef>` attribute.

Knowing that the process definition and instance management is done by humans who might not have the technical capability to deal with the low-level language described in this chapter, we decided to develop a web visual interface using well known interactive paradigms.

5.3. User Interface for Process Management

In this section we describe the tools used, how we used them, what decisions we have taken and why, to develop a web user interface for process orchestration, from the definition of a process through a visual programming paradigm, to the custom input for each task, and the runtime monitoring of process instances.

5.3.1. Google Web Toolkit as Web Development Framework

After an analysis to web development frameworks (section 2.5) we concluded that GWT was the right one for us. Based on the GWT architecture, our prototype's architecture is also composed of a server side responsible by retrieving the information from the sources outside the system and sending it to the client side, where it is processed and showed in the user's browser. This type of architecture allows for a good performance, saving bandwidth for data exchange only, and being UI loading done on client side. Also it provides an easy deployment and cross-browser support. To allow a more attractive *look and feel*, we also used an *open-source* GWT widget extension named Ext-GWT. This enabled us to create the web interface faster using Ext-GWT's widgets, which can be easily change according to the user's needs.

5.3.2. Process Definition through a Visual Programming Paradigm

To visually define each process, we used an extension of the BPMN 2.0 visual notation based on workflows (further details on section 2.1.3). The user is presented with a set of possible components (each with its own visual representation) that can be added to the process (Figure 19 - bottom left side), and connected with arrows (sequence flows) to define the workflow execution sequence.

On the middle of the screen, the user is presented with a design grid where the components can be placed and moved through *drag and drop* from the "Shape Repository" (Figure 19 - left side). When a component is selected, its properties are displayed depending on the type of component (Figure 19 - right side). There are properties that every component has like "Name" and "Position", but some components have specific properties that are used as input to execute that component. As shown in the previous figure, to execute the harvest component, the user has to select the identifier of the data source to be harvested; choose to send an email when the harvest is complete; and if it is a full or partial harvest. This kind of integration between process modelling and UI custom input for each component create a seamless experience during process definition, enhancing the user's performance (as described in section 2.4).

This modelling interface was developed using a Scalable Vector Graphics (SVG)[71] library for GWT developed by us. This library provides the necessary integration between SVG objects and GWT, allowing us to create a SVG drawing board within a GWT panel (Figure 19 – center). Through this drawing board we are able to know how many SVG components are within the board, and also their properties and position. This information allowed us to transform the process' visual SVG components to the extended BPMN 2.0 XML format described on section 5.2.2.1, permitting a preservation of the workflow sequence, visual design information and each component's parameters through their visual properties.

This approach creates a seamless environment for process definition that allows the user to not only create a workflow sequence using the given component set, but also insert the input of each component through a custom visual interface that depends on the selected component. After the visual definition a file is created (through the schema described in section 5.2.2.1) which provides persistency to the process, enabling future editing and the creation of process instances ready to be executed using the *jBPM* engine.

[71] Scalable Vector Graphics (SVG) - http://www.w3.org/Graphics/SVG/

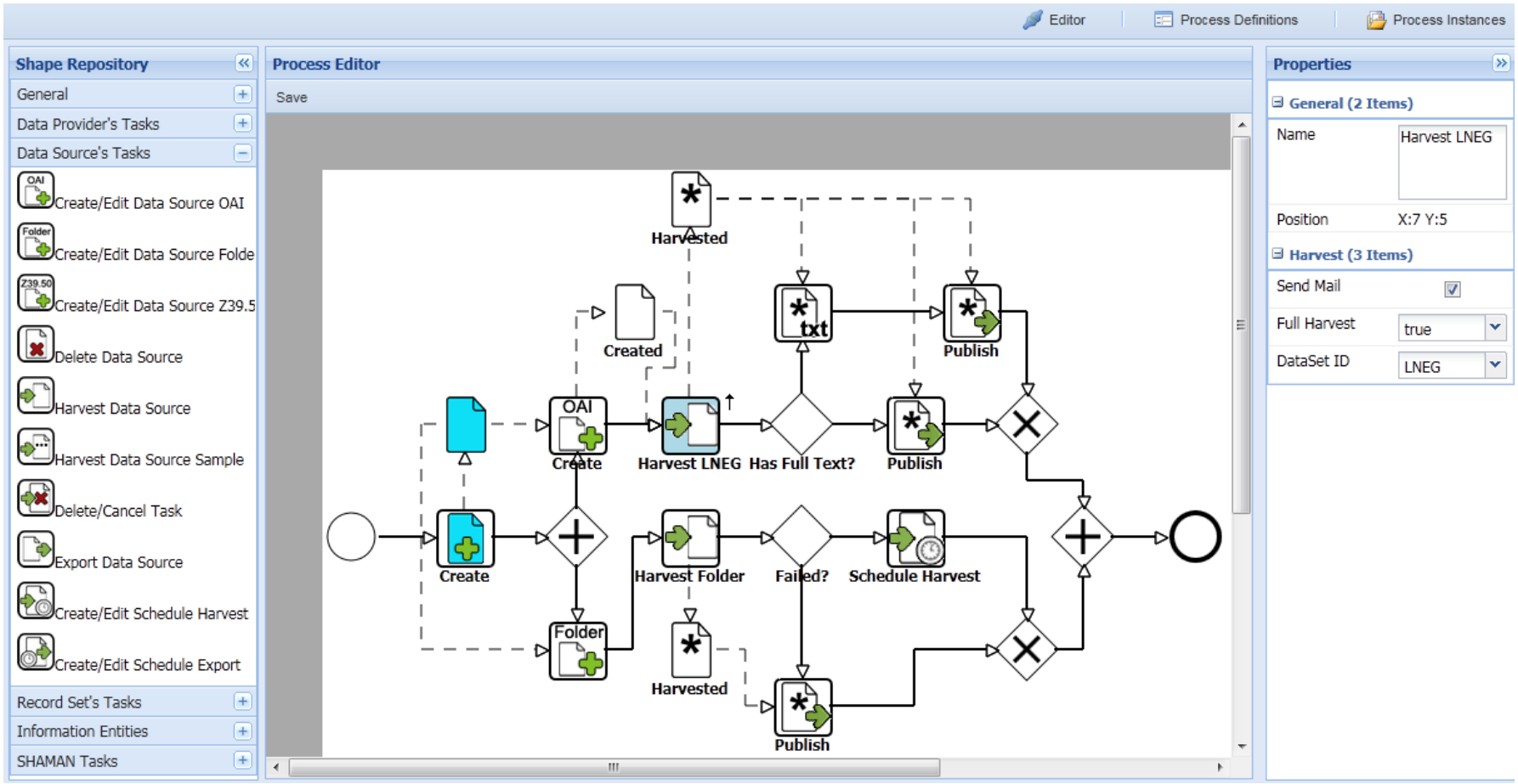


Figure 19. Modeling of a process using our process definition editor.

5.3.3. Process Definitions Management

Being confronted with the possibility of having several process definitions in the system, we needed some interface that enabled the user to manage all of them in a simple and natural way. Therefore, and based on a set of researched management user interfaces like Knoodle's presentation management UI[72] and jBPM's web console (Figure 11), we developed a simple interface that shows the available process definitions on the system (Figure 20).

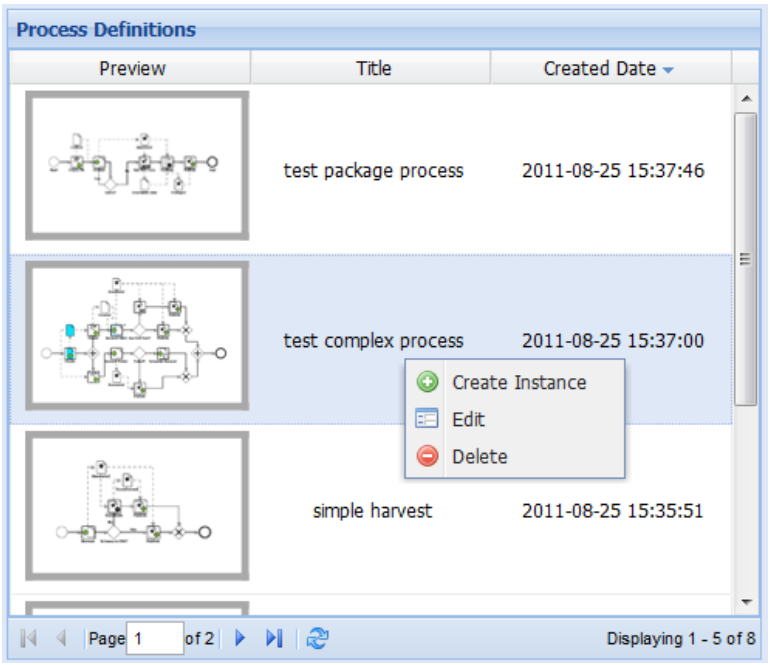


Figure 20. Process definition management UI.

The information of each process shown in the previous figure (creation date, title and SVG preview) is saved in a XML that contains not only this information, but also a reference to each process definition file name. This way we create a main XML file with all the process definitions in the system with references to each individual process definition file (An example of a system's process definitions XML file can be seen in Appendix E). As a result, through these references the user is able to edit existing process definitions and create new instances by right-clicking on the process definition and choosing the operation from the menu that appears (Figure 20). Finally, if the user creates a new instance, a new process instance XML file is created using the schema described in section 5.2.2.2, and is added to the main file containing all process instances in the system (explained in detail in the next section).

[72] Presentation Management UI - http://www.knoodle.com/about/media-kit

5.3.4. Process Instance Management and Monitoring

Similar to the problem we faced in the previous chapter, we knew that multiple instances could be created through a process definition, and so we would need another interface capable of dealing with the management of several process instances. The difference from using the previous approach is that we knew that this interface had to be capable of making the user's job easier during the monitoring of all the process instances running at the same time. As a result we came up with a possible solution where the user can watch several executing instances and their current state (Figure 21).

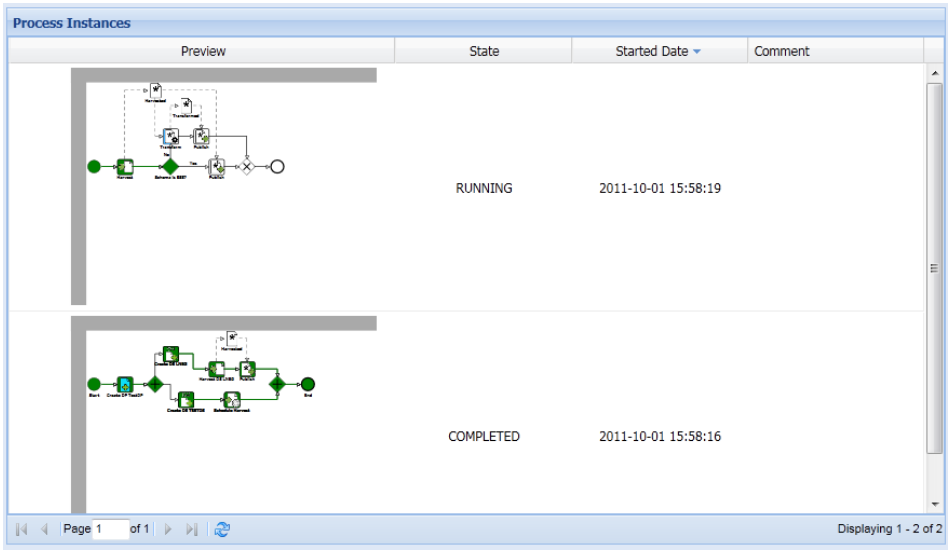


Figure 21. Process instance management UI.

Like the previous approach for process definition management, we also use a XML file to represent all the process instances in the system with references to each process instance file (An example of a system's process instances XML file can be seen in Appendix F). Also, the user can right-click on a process instance and choose between the following operations:

- View Instance – View more detailed information about a single instance (Figure 23);
- Start – Start a process instance;
- Stop – Stop a process instance;
- Set Comment – Set the comment for a process instance displayed in the main overview table shown in Figure 21 (This can be used by the user to describe the purpose of different process instances);
- Delete – Delete a process instance.

This solution provides the user a method to constantly monitor and manage each process instance in the system, simply by navigating through the table and verify the instance's

components colors to determine the current status of execution (further details about the color's meaning will be explained in the next section). As a result, the user performance in process monitoring and management is increased [24].

5.3.5. Component State Color Glossary

To represent the current state of a component within a process, we use a set of different colors and stroke patterns. Therefore, according to the state of a given component (described in section 5.2.2.1) its visual background color changes. Some examples of using colors to represent different states are given below (Figure 22).

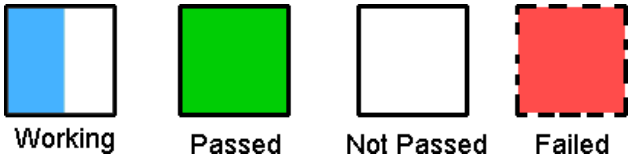


Figure 22. Component state color representation.

Such approach enables process instance monitors to check quickly what the current status of the process is, increasing its performance [24] (An example of the usage of this technique can be seen in Appendix C).

5.3.6. Process Instance Detailed Monitoring

To understand some problems occurring during an instance's execution, and also retrieve a better knowledge of what's happening, more detailed information about the executing instance must be presented to the user. As a result, we created a more complete view that shows a single process instance in its current state (Figure 23).

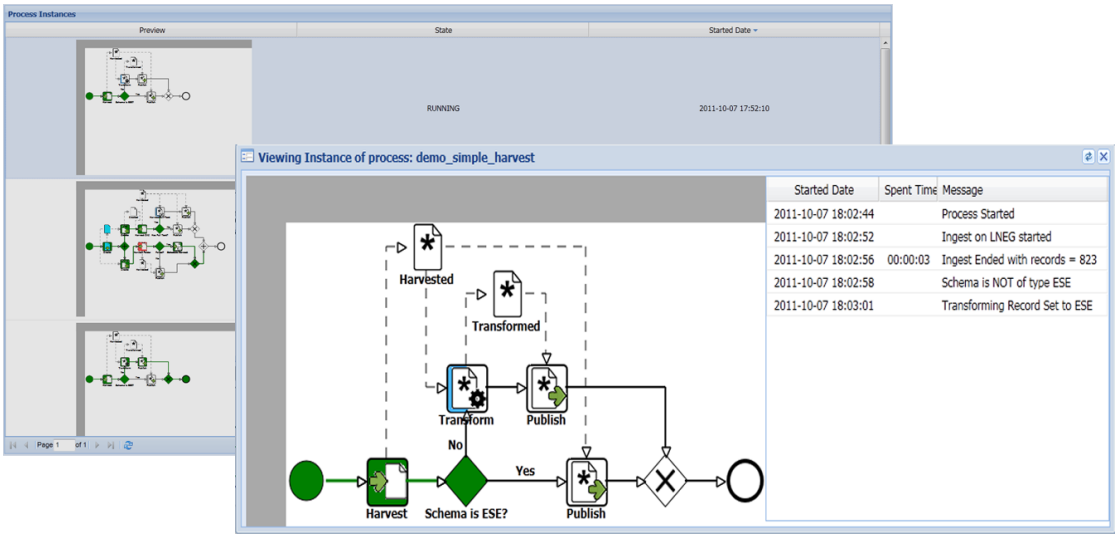


Figure 23. Detailed view of a parallel ingest process.

For this more thorough view presented in the previous figure, an up scaled process execution graphic is presented in the left side, completed with the help of an execution log (created using the schema described in section 5.2.2.2) on the right side. This provides a correlation between the visual runtime feedback of the process instance graphic and the messages being sent by each component, along with the time they are taking to complete, thus providing an improved runtime analysis.

5.4. Summary

In this chapter, we have seen the full extension of the proposed architecture, with detailed information about each of its components.

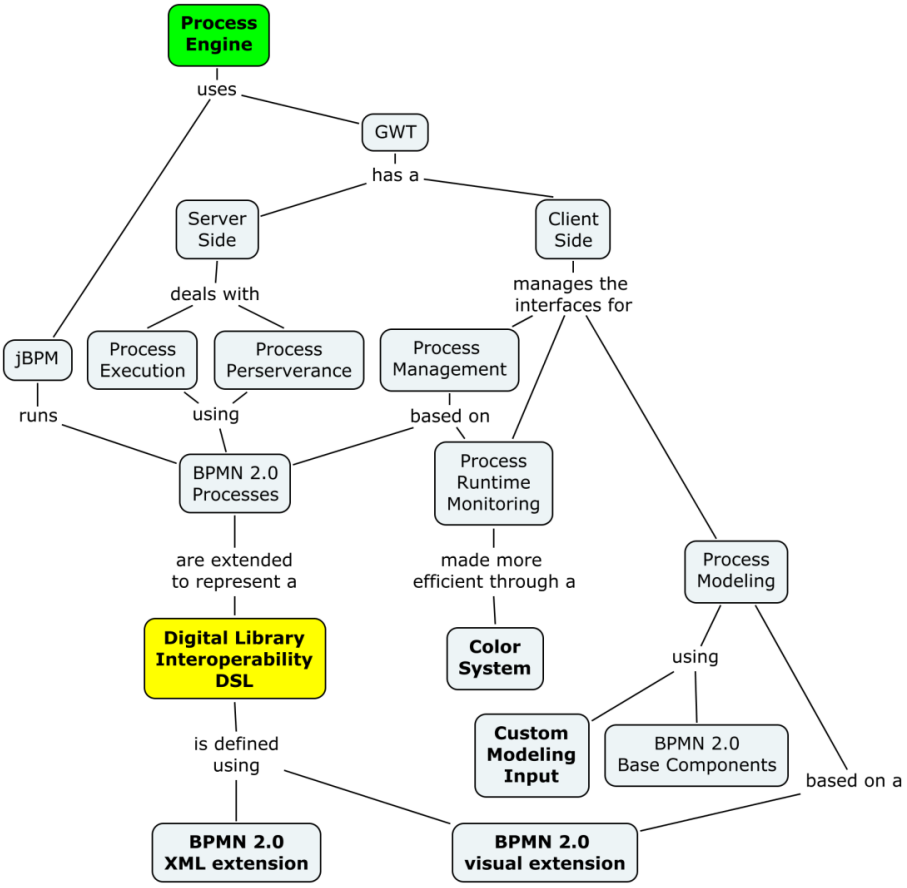


Figure 24. Proposed solution concept model.

As described in Figure 24, the proposed solution for our process engine uses the jBPM and GWT frameworks. The first one is used to run our extended BPMN 2.0 processes that represent the proposed DSL described in chapter 4, and are defined through a BPMN 2.0 XML schema and visual notation extension. The GWT framework is separated in a server side that communicates with the jBPM and therefore manages process execution and perseverance. On the other hand, the GWT's client side manages the interfaces used for process modeling (where some BPMN 2.0 base components are used and several interface techniques for custom input during modeling are applied), process definition and instance management and process runtime monitoring (made more efficient through a color and pattern technique to represent each component's execution state.

The next chapter will start by detailing the validation of our solution.

6

Evaluation

In this chapter we evaluate our solution through the execution and monitoring of processes based on our DSL, and applied to real scenarios related to digital library interoperability. This evaluation is accomplished by running with simple and complex processes to test the full capability of our solution. Finally, we evaluate the extensibility of our solution through its capability to easily adapt to new tasks and behaviors.

6.1. Methodology

The idea for the verification of the hypothesis was to apply the developed DSL-based prototype to the REPOX framework used in various scenarios related with Europeana, TEL, EuDML and SHAMAN. The main purpose is to evolve the currently used "hard-coded" architecture into a service oriented architecture with a web human interface for process orchestration, and as a result, provide more flexibility using XML for process representation and improve user performance on process modeling and monitoring [24] through the use of a DSL, which is easier to use by domain experts [6].

6.1.1. Simple Process Evaluation

For the first evaluation we created a simple process that harvests the data of the UniversidadeAberta record set from a REPOX instance[73]. Then, according to the schema of the harvested record set, it is transformed or not to the ESE schema. According to the decision, the originally harvested record set or the transformed one is published on the REPOX database in order to make it available by OAI-PMH. (the XML definition of this process can be found on Appendix B)

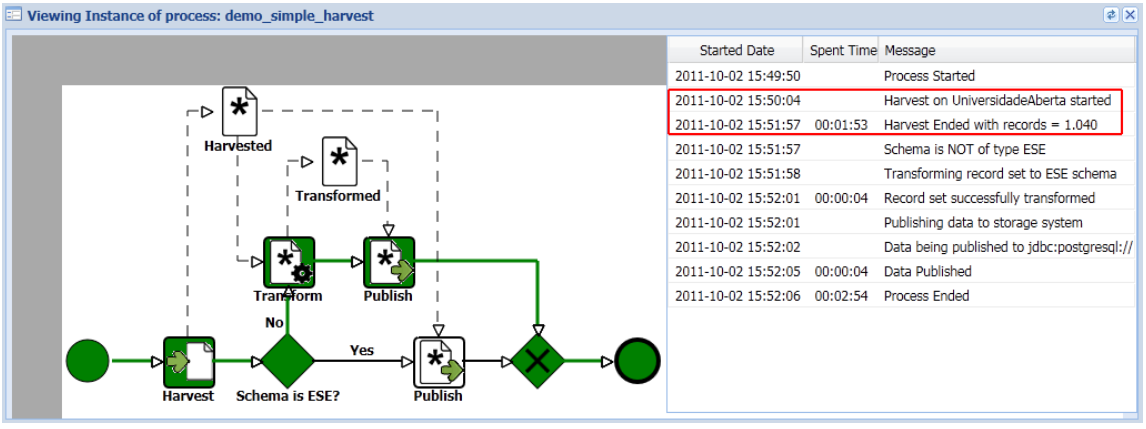


Figure 25. Simple harvest process.

[73] REPOX instance used for testing located in http://bd2.inesc-id.pt:8080/repoxLight

The previous Figure 25 illustrates our framework running the simple harvest process, in the condition that the record set harvested is not of schema type ESE. To verify the process' success or failure, we logged into the REPOX web interface to verify the last ingest of the UniversidadeAberta record set, the record count, the time of the harvest, and finally the schema of the record set (Figure 26).

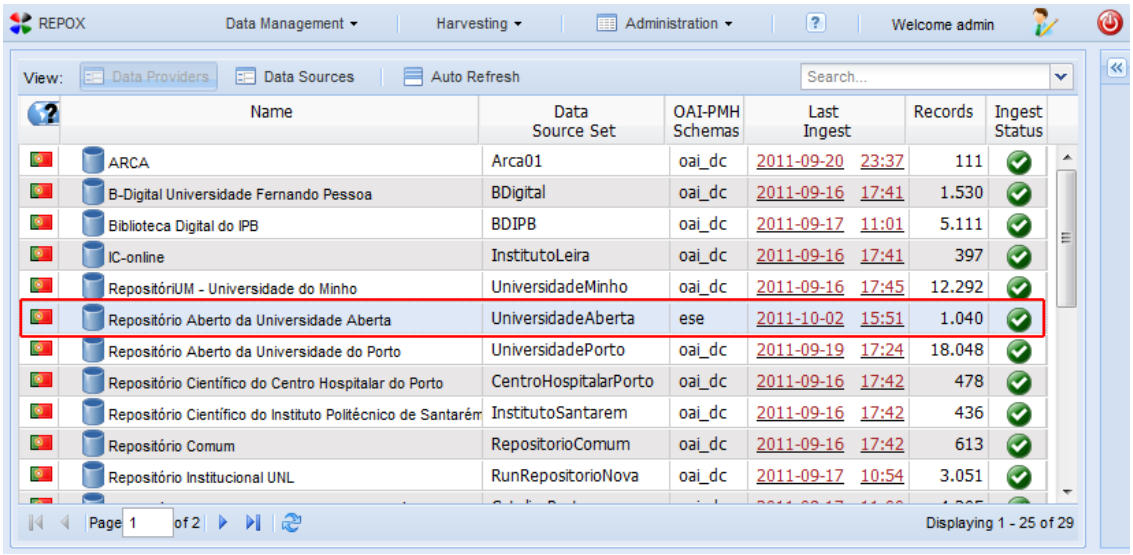


Figure 26. REPOX web interface used by the Europeana project with harvested data of the UniversidadeAberta record set.

To test more thoroughly our solution, we created more a complex process that further enhances the importance of data exchange between a process' tasks and other DSL operations that can be used in a process.

6.1.2. Complex Process Evaluation

As an example of a complex process (Figure 27), we decided to define a process that contains the full cycle of data harvesting, from the creation of the data provider and their data sources, to their harvest and publishing, taking into account the success/failure of one of the harvests, and the existence of full-text on one of the harvested record sets.

The process represented in Figure 27 begins by creating the base storage structures for data harvest (a data provider with two data sources of different types – OAI and Folder). This data source creation is done in parallel, which allows us to on one side, harvest the data from the OAI data source and then, after verifying the existence of full-text on the harvested record set, harvest it as well. On the other side, and since we are dealing with a system where unexpected problems can occur, we simulate a harvest failure which we deal with by scheduling it to a future date.

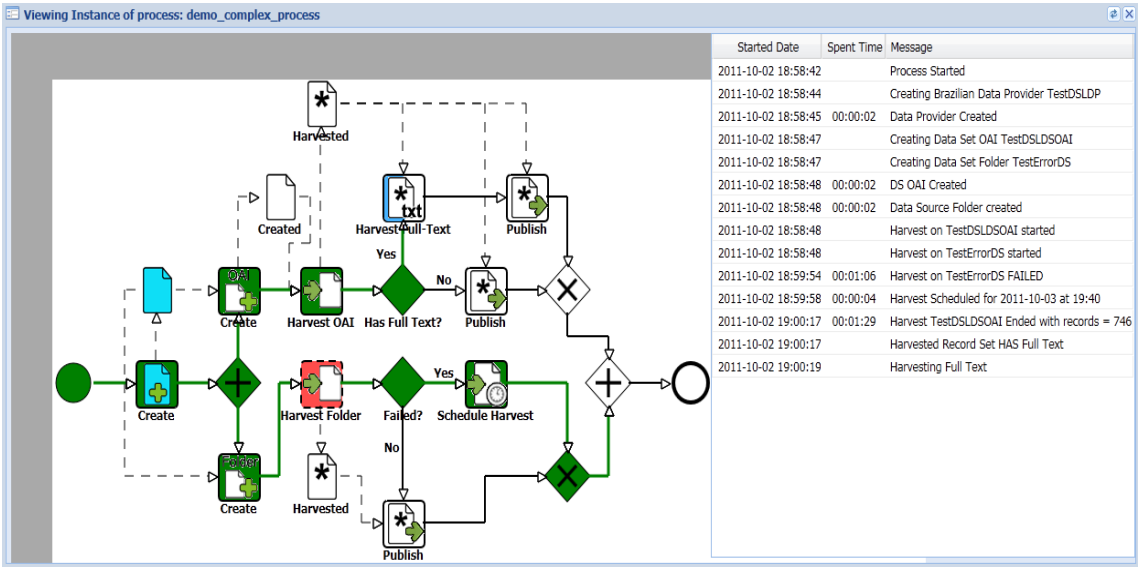


Figure 27. Complex process runtime monitoring.

As shown in Figure 27, the running process shows detailed information on the right side log grid about each task's data, like for example the chosen name (*TestDSLDP*) and country (*Brazil*) of the data provider, the names of the created data sources (*TestDSLDSOAI* and *TestErrorDS*), harvested record count (*746*), scheduled harvest date and time (*03/10/2011 at 19:40*), the *TestErrorDS* ingest status, and all the task's start dates and spent times. Finally, analyzing the process' log data and the REPOX's web interface (Figure 28) we can confirm that the data obtained during the process' execution corresponds to the one represented in REPOX, and therefore showing the successful integration between our solution and the REPOX framework.

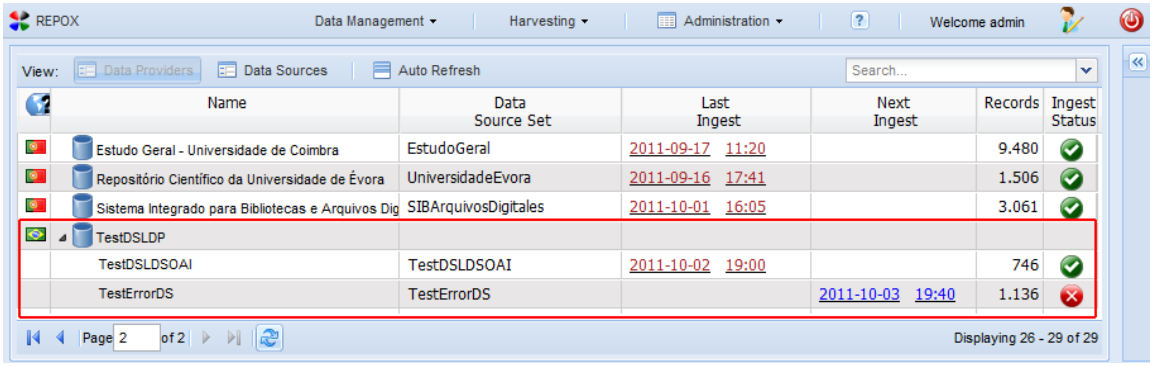


Figure 28. REPOX web interface used in the TEL project after the previously described complex process' execution.

To evaluate the extensibility of our solution to new requirements on digital library interoperability, we extended our solution to the SHAMAN project, adding new tasks according to the project's requirements.

6.1.3. SHAMAN - Extensibility to new types of tasks and behaviors

The SHAMAN project addresses the scope of digital preservation. Its purpose is to propose new concepts and techniques for digital libraries with capabilities for preservation (these systems are more commonly named "archives", than "libraries").One example addressed in the SHAMAN project is to harvest data from the *myExperiment*[74] system and package it in new data objects properly structured for preservation. To be able to cope with the new requirements we had to develop three new tasks:

- *Create Data Source ME* – A data source for specifically harvesting the data from the *myExperiment* web service system;
- *MDR* – A metadata registry manager that, using a given record set, creates descriptive metadata about it;
- *Create Descriptive Package* – A service that packages a harvested record set and its descriptive metadata into another record set.

Using the DSL we proposed and these two new tasks, we were able to create a process that harvests data from the *myExperiment* server, creates structured descriptive metadata about it, packages the result and publishes it in the SHAMAN database to be shared.

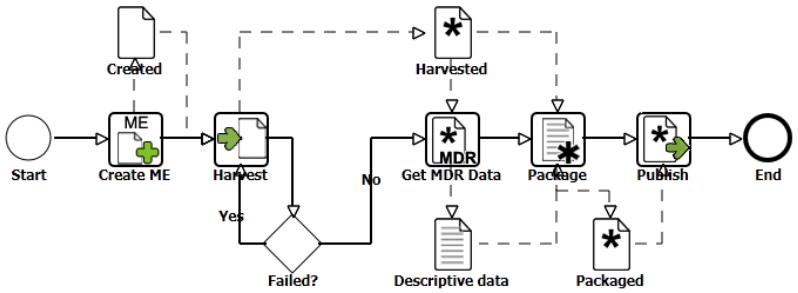


Figure 29. SHAMAN process for data harvest.

This example demonstrates that it is possible to extend the capabilities of the DSL and the framework to harvest additional types of information and perform new operations simply by adding new tasks (their DSL visual and semantic component) with specific behavior (the corresponding web service and data input and output type).

[74]myExperiment - http://www.myexperiment.org/

6.2. Summary

The results from this study suggest that it is possible to apply a DSL supported by a web services architecture to the Tel, Europeana and SHAMAN scenarios each supported by the REPOX metadata interchange framework. Also, users familiar with the digital library interoperability domain can more efficiently model business processes if they are using domain specific concepts [6].

Testing the simple and complex processes we realized that tasks defined through web services allow the use of several operations (like the *Harvest*, *Transform*, *Publish Record Set*, etc.), and using information entities for data exchange between tasks provide a flexible system for process definition. Also, supported by a color and pattern technique to represent each task's state, and a log table constantly updated while the process is executing, provides important feedback to the user while monitoring each process instance [24].

Finally, in the SHAMAN scenario where some requirements were not met by our initial DSL, we proved that by simply adding some new tasks (their DSL visual and semantic component) with specific behavior (the corresponding web service), a new harvest process can be defined using the new data.

In the next chapter, we will finalize this dissertation by wrapping up what has been said until here, clarify the contributions and limitations of this work, and state what can be done in the future.

7

Conclusions

7. Conclusions

Although domain-specific languages (DSLs) are costly to design and implement, and require the learning of a new language with limited capability (only applied to its domain), they allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL programs. As a result, a good DSL can enhance quality, productivity, portability and reusability [23].

In the Related Work section, we learned the importance of BPM in an organization's efficiency, effectiveness and flexibility to adapt to changes. Its visual representation (BPMN) is tailored for creating graphical models of business process operations, and therefore creates a standardized bridge for the gap between the business process design and process implementation [40]. The current version of BPMN (specification 2.0)[11] not only defines a standard on how to graphically represent a business process like BPMN 1.x, but also includes execution semantics for the elements defined, and an XML format on how to store process definitions. As a result, this made it the best suitable candidate for the implementation of our DSL by extending the BPMN 2.0.

In order to find some guidelines on how to design a DSL, we researched some solutions presented by Steen Brahe et al. [6] and Momotko [24] which show a set of guiding principles used when defining a DSL, and present techniques for DSL creation based on BPMN, through the use of colors and custom icons. Also, to find a suitable approach to define and execute our DSL, we studied some workflow technologies and concluded that solutions like *jBPM* and *Activiti,* which use BPMN 2.0 as the core process definition and execution language to their process engine, have the flexibility to define complex processes. However, they still lack the process execution monitoring interfaces and modeling input interfaces, which are important issues for process managers [26][4].

Finally, to create our DSL-based web framework for process orchestration we searched for the right tool for the job. After comparing some web development frameworks we concluded that GWT, though it has a medium learning curve and ease of use, it provides a high performance and extensible framework to build complex UI interactions, eventually being the more fitting to our problem.

We analyzed the problem and realized that each institution (Libraries, Museums, and Archives) uses different standards for their metadata representation, which makes it hard to create interoperability between them, in order to accomplish an easy exchange of data for a common use. Through the use of emerging standards new approaches to metadata interoperability became possible. However, an important issue will always be the aggregation of those metadata sets from their original sources, which in some cases can have serious

scalability requirements[75]. A fundamental piece to address that issue has been the OAI-PMH data exchange protocol. Based on the OAI-PMH, several projects like REPOX started to develop new frameworks to automatically manage metadata harvesting. Still, new emerging scenarios for transfer not only of data sets but also the contents referenced by these data sets (for example, the harvesting of the full-text of the documents described in the data sets) require the support for more sophisticated harvesting and aggregation processes.

In order to orchestrate these new processes with enough flexibility, we started by designing our domain specific language for digital library interoperability. This design was based on some DSL creation principles that recommend the definition of the tasks and their data types in a DSL, and some visual programming techniques for the visual component in our DSL. Also, based on the goals and requirements obtained while analyzing the problem, we defined the main concepts in digital library interoperability, which lead us to create a domain concepts glossary that shows the description and usage of standard visual symbols to represent the *Information Entities*, *Operations* and *Technology* concepts of our DSL. Supported by these domain concepts, we finally define the tasks of our DSL.

To implement our DSL we used an integrated solution between the jBPM and GWT frameworks to create a process engine with a human interface. The first one is used to run our extended BPMN 2.0 processes that represent the Proposed DSL described in chapter 4, and are defined through a BPMN 2.0 XML schema and visual notation extension. The GWT framework is separated in a server side that communicates with the jBPM and therefore manages process execution and perseverance. On the other hand, the GWT's client side manages the interfaces used for process modeling (where some BPMN 2.0 base components are used and several interface techniques for custom input during modeling are applied), process definition and instance management and process runtime monitoring (made more efficient through a color system for each component's state representation.

Finally, the evaluation results suggest that it is possible to apply a DSL supported by a web services architecture to the Tel, Europeana and SHAMAN scenarios each supported by the REPOX metadata interchange framework. Also, users familiar with the digital library interoperability domain can more efficiently model business processes if they are using domain specific concepts [6].

In general, we were able to create a DSL that enables digital libraries interoperability when integrated with a web service architecture for process orchestration. Our proposed solution:

[75]The catalogues of most national libraries have records in the order of the millions.

- Eliminates the need for technical knowledge in the creation of business processes through a process modeler based on a BPMN 2.0 extended visual notation;
- Increases user performance in process monitoring through a color and pattern system used to represent a process' tasks state;
- Increases flexibility in process exchange and through the use of a persistent format like XML to represent each process;
- Is extensible enough to easily create new tasks, with each task representing a web service handler, and its actual behavior created through the definition of this handler, its visual notation, and output and input data;
- Enables the application of complex transformations to the harvesting processes.

Overall, our results suggest that our solution is a valid approach to support digital library interoperability, at least in real scenarios like Tel, Europeana, EuDML and SHAMAN.

7.1. Future Work

One of the limitations of our work is the use of bitmap icons instead of SVG. Although all the process modeler's components are SVG, the small icons inside each task are bitmap, which limits the capability of the whole process being defined in SVG, preventing the usage of third-party SVG tools to reproduce the process, and even restricting its own visual quality. Therefore in the future, we would like to fully convert the components to SVG.

The process modeler also should provide additional BPMN primitives like Human Tasks. These kinds of tasks, as the name suggests, only complete after human order. They are important to define scenarios where a human decision must be taken within a process.

Although now we provide support to define complex transformations in the harvesting processes of the current REPOX version (2.0), our goal is to extend the use of our processes to all processes in REPOX, and enable the seamless exchange of process definitions between REPOX instances.

Also new visual notation approaches like Momotko [24] should be considered, in which additional runtime information is inserted directly on the task's visual notation, and therefore enhancing process monitoring. Finally, we should promote user testing to further evaluate the web interface's ease of use on process monitoring and modeling.

References

. References

[1] J. R. Ahronheim. Descriptive metadata: Emerging standards. volume 24, pages 395–404. Journal of Academic Librarianship.

[2] Cyrille Artho, Klaus Havelund, and Shinichi Honiden. Visualization of concurrent program executions. In *Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 02*, COMPSAC '07, pages 541–546, Washington, DC, USA, 2007. IEEE Computer Society.

[3] Janis Barzdins, Karlis Cerans, Mikus Grasmanis, Audris Kalnins, Sergejs Kozlovics, Lelde Lace, Renars Liepins, Edgars Rencis, Arturs Sprogis, and Andris Zarins. Domain specific languages for business process management: a case study. *The 9th OOPSLA Workshop on Domain-Specific Modeling*, 2009.

[4] Stefan Betermieux and Birgit Bomsdorf. Finalizing dialog models at runtime. In Luciano Baresi, Piero Fraternali, and Geert-Jan Houben, editors, *Web Engineering*, volume 4607 of *Lecture Notes in Computer Science*, pages 137–151. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-73597-7_11.

[5] S.D. Bragg and C.G. Driskill. Diagrammatic-graphical programming languages and dod-std-2167a. In *AUTOTESTCON '94. IEEE Systems Readiness Technology Conference. 'Cost Effective Support Into the Next Century', Conference Proceedings.*, pages 211 –220, sep 1994.

[6] Steen Brahe and Kasper Østerbye. *Business process modeling: Defining domain specific modeling languages by use of uml profiles*, pages 241–255. Springer, 2006.

[7] Henri Chen and Robbie Cheng. *ZK: Ajax without the Javascript Framework*. Apress, Berkely, CA, USA, 2007.

[8] Dario Correal and Rubby Casallas. Using domain specific languages for software process modeling. *Universidad de los Andes*, 2007.

[9] Renata Queiroz Dividino, Veli Bicer, Konrad Voigt, and Jorge Cardoso. Integrating business process and user interface models using a model-driven approach. In *ISCIS*, pages 492–497. IEEE, 2009.

[10]Nuno Freire and José Borbinha. Metadata spaces: The concept and a case with repox. In *Research and Advanced Technology for Digital Libraries*, volume 4172 of *Lecture Notes in Computer Science*, pages 516–519. 2006.

[11]J.J. Garrett et al. Ajax: A new approach to web applications. 2005.

[12]M. Giorgi. Comparison of ajax frameworks: Prototype, gwt, dwr and thinware. 2007.

[13]John Grundy, John Hosking, Nianping Zhu, and Na Liu. Generating domain-specific visual language editors from high-level tool specifications. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 25–36, Washington, DC, USA, 2006. IEEE Computer Society.

[14]W. Han, L. Di, P. Zhao, and X. Li. Using Ajax for desktop-like geospatial web application development. In *Geoinformatics, 2009 17th International Conference on*, pages 1–5. IEEE, 2009.

[15]Marta Indulska, Jan Recker, Michael Rosemann, and Peter Green. Business process modeling: Current issues and future challenges. In *Proceedings of the 21st International Conference on Advanced Information Systems Engineering*, CAiSE '09, pages 501–514, Berlin, Heidelberg, 2009. Springer-Verlag.

[16]Federico Kereki. Web 2.0 development with the google web toolkit. *Linux J.*, 2009(178):2, 2009.

[17]M. Kloppmann, D. König, F. Leymann, G. Pfau, and D. Roller. Business process choreography in websphere: combining the power of bpel and j2ee. *IBM Syst. J.*, 43(2):270–296, 2004.

[18]Carl Lagoze and Herbert Van de Sompel. The open archives initiative: building a low-barrier interoperability framework. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 54–62, New York, NY, USA, 2001. ACM.

[19]Joe Lennon. Compare javascript frameworks. IBM Press, 2010.

[20]Chengfei Liu, Qing Li, and Xiaohui Zhao. Challenges and opportunities in collaborative business process management: Overview of recent advances and introduction to the special issue. *Information Systems Frontiers*, 11:201–209, July 2009.

[21]Ryan McFall and Charles Cusack. Developing interactive web applications with the google web toolkit. *J. Comput. Small Coll.*, 25(1):30–31, 2009.

[22]Katharina Mehner. Javis: A uml-based visualization and debugging environment for concurrent java programs. In *Revised Lectures on Software Visualization, International Seminar*, pages 163–175, London, UK, 2002. Springer-Verlag.

[23]Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37:316–344, December 2005.

[24]Mariusz Momotko and Bartosz Nowicki. Visualisation of (distributed) process execution based on extended bpmn. In *Proceedings of the 14th International Workshop on Database and*

Expert Systems Applications, DEXA '03, pages 280–, Washington, DC, USA, 2003. IEEE Computer Society.

[25]Michael Zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, CAiSE '08, pages 465–479, Berlin, Heidelberg, 2008. Springer-Verlag.

[26]Michael zur Muehlen and Michael Rosemann. Workflow-based process monitoring and controlling - Technical and organizational issues. In *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6 - Volume 6*, HICSS '00, pages 6032–, Washington, DC, USA, 2000. IEEE Computer Society.

[27]Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20:3045–3054, November 2004.

[28]Linda Dailey Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, 2005.

[29]Liu Peng and Bosheng Zhou. Research on workflow patterns based on jbpm and jpdl. In *PACIIA '08: Proceedings of the 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, pages 838–843, Washington, DC, USA, 2008. IEEE Computer Society.

[30]Jan C. Recker. Bpmn modeling – who, where, how and why. *BPTrends*, 5(3):1–8, March 2008.

[31]Marie-Christine Rousset and Chantal Reynaud. Knowledge representation for information integration. volume 29, pages 3–22. Elsevier Science Ltd., Oxford, UK, 2004.

[32]Stefanie Ruehle, José Borbinha, and Gilberto Pedrosa. *Deliverable 4.1 - Requirements Infrastructure and Harvester*. Europeana Libraries: Aggregating digital content from Europe's libraries, 2011.

[33]Nicolás Serrano and Juan Pablo Aroztegi. Ajax frameworks in interactive web apps. *IEEE Softw.*, 24(5):12–14, 2007.

[34]Mark Strembeck and Uwe Zdun. An approach for the systematic development of domain-specific languages. *Softw. Pract. Exper.*, 39:1253–1292, October 2009.

[35]Hussein Suleman. Introduction to the open archives initiative protocol for metadata harvesting. In *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 414–414, New York, NY, USA, 2002. ACM.

[36]Ian Taylor, Matthew Shields, Ian Wang, and Roger Philp. Grid Enabling Applications Using Triana. In *Workshop on Grid Applications and Programming Tools*. Held in Conjunction with GGF8, 2003.

[37]Hallvard Trætteberg and John Krogstie. Enhancing the usability of bpm-solutions by combining process and user-interface modelling. In *The Practice of Enterprise Modeling*, volume 15 of *Lecture Notes in Business Information Processing*, pages 86–97. Springer Berlin Heidelberg, 2009.

[38]W. M. P. van der Aalst and A. H. M. ter Hofstede. Yawl: yet another workflow language. *Inf. Syst.*, 30:245–275, June 2005.

[39]Stuart L. Weibel and Carl Lagoze. An element set to support resource discovery. *International Journal on Digital Libraries*, 1:176–186, 1997.

[40]S. White. Using BPMN to model a BPEL process. *BPTrends*, 3(3):1–18.

[41]S.A. White. Introduction to BPMN. *IBM Cooperation*, pages 18–29, 2004.

[42]Dave Fulker Carl Lagoze William Y. Arms, Naomi Dushay. A case study in metadata harvesting: the nsdl. In *Library Hi Tech*, volume 21, pages 228–237, 2003.

[43]Shaohua Xie, Eileen Kraemer, R. E. K. Stirewalt, Laura K. Dillon, and Scott D. Fleming. Design and evaluation of extensions to uml sequence diagrams for modeling multithreaded interactions. *Information Visualization*, 8:120–136, April 2009.

[44]Nianping Zhu, John Grundy, and John Hosking. Pounamu: A meta-tool for multi-view visual language environment construction. In *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, VLHCC '04, pages 254–256, Washington, DC, USA, 2004. IEEE Computer Society.

Appendices

A. DSL Manual

a. DSL execution semantics for Data Provider's tasks

Task	<Attribute>/Type and Return
Create/Edit Data Provider	- **<Attribute>/Type**: <Name>/String, <2 Letters Country>/String, <Description>/String, <IsUpdate>/Boolean, <Provider Reference>/String[76] - **Return**: <Data Provider>/Data Provider
Delete Data Provider	- **<Attribute>/Type**: <Provider Reference>/String - **Return**: void

b. DSL execution semantics for Data Sources' tasks

Task	<Attribute>/Type and Return
Create/Edit Data Source – OAI	- **<Attribute>/Type**: <Provider Reference>/String, <Datasource Reference>/String, <Name>/String, <Description>/String, <Export Path>/String, <Schema>/String, <Namespace>/String, <Metadata Format>/String, <OAI Server URL>/String, <OAI Set>/String, <IsUpdate>/Boolean - **Return**: <Data Source>/Data Source
Create/Edit Data Source - Z39.50	- **<Attribute>/Type**: <Provider Reference>/String, <Datasource Reference>/String, <Name>/String, <Description>/String, <Export Path>/String, <Schema>/String, <Namespace>/String, <Address>/String, <Port>/String, <Database Name>/String, <User>/String, <Password>/String, <Record Syntax>/String, <Charset>/String, <Namespace Prefix>/String, <Namespace Uri>/String, <Record Id Policy>/String, <ID Xpath>/String, <Z39.50 Type>/String, <Earliest Time Stamp(AAAAMMDD)>/String[77], <File Path with ID list>/String[77], <Maximum ID>/String[77], <IsUpdate>/Boolean - **Return**: <Data Source>/Data Source
Create/Edit Data Source – Folder	- **<Attribute>/Type**: <Provider Reference>/String, <Datasource Reference>/String, <Name>/String, <Description>/String, <Export Path>/String, <Schema>/String, <Namespace>/String, <Metadata Format>/String, <ISO Format>/String, <Charset>/String, <Namespace Prefix>/String, <Namespace Uri>/String, <Record Id Policy>/String, <ID Xpath>/String, <Record Xpath>/String, <FType>/String, <Server Url>/String[78], <User>/String[78], <Password>/String[78], <FTP Directory Path>/String[78], <Url>/String[78], <Folder Path>/String[78], <IsUpdate>/Boolean

[76] The field is required according to the IsUpdate field and optional otherwise.

[77] The field is required according to the Z39.50 Type field and optional otherwise.

[78] The field is required according to the FType field and optional otherwise.

	- **Return**: <Data Source>/Data Source
Delete Data Source	- **<Attribute>/Type**: <Datasource Reference>/String - **Return**: void
Harvest Data Source	- **<Attribute>/Type**: <Datasource Reference>/String - **Return**: <Record Set>/Record Set
Harvest Data Source Sample	- **<Attribute>/Type**: <Datasource Reference>/String - **Return**: <Record Set>/Record Set
Delete/Cancel Task	- **<Attribute>/Type**: <Datasource Reference>/String - **Return**: void
Export Data Source	- **<Attribute>/Type**: <Datasource Reference>/String, <Records Per File>/String - **Return**: void
Create/Edit Schedule Harvest	- **<Attribute>/Type**: <Datasource Reference>/String, <Start Date>/Date, <Period>/String, <Full Ingest>/Boolean,<IsUpdate>/Boolean,<Schedule ID>/String[76] - **Return**: void
Create/Edit Schedule Export	- **<Attribute>/Type**: <Datasource Reference>/String, <Start Date>/Date, <Period>/String, <Full Ingest>/Boolean,<IsUpdate>/Boolean,<Schedule ID>/String[76] - **Return**: void

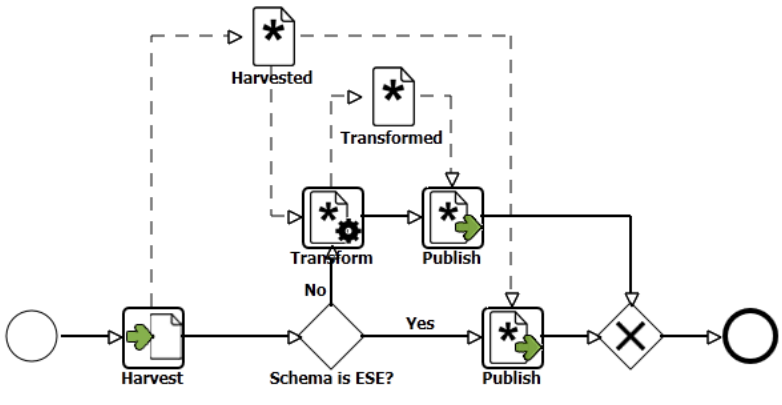
C. DSL execution semantics for Record's tasks

Task	<Attribute>/Type
Save Data Record	- **<Attribute>/Type**: <Data Record ID>/String, <Datasource Reference>/String, <Record String>/String - **Return**: <Data Record>/Data Record
Delete Data Record	- **<Attribute>/Type**: <Data Record ID>/String - **Return**: void
Publish Record Set	- **<Attribute>/Type**: <Set>/Record Set - **Return**: void
Get Record Set	- **<Attribute>/Type**: <Data Source ID>/String - **Return**: <Record Set>/Record Set
Transform Record Set	- **<Attribute>/Type**: <Set>/Record Set - **Return**: <Record Set>/Record Set
Package Record Set	- **<Attribute>/Type**: <Records List>/List< Data Record>, <Target Record Set>/Record Set - **Return**: <Record Set>/Record Set
Harvest Data Record Full-Text	- **<Attribute>/Type**: <Record>/Data Record - **Return**: void
Harvest Record Set Full-Text	- **<Attribute>/Type**: <Set>/Record Set - **Return**: void

B. Example of a simple harvest process definition

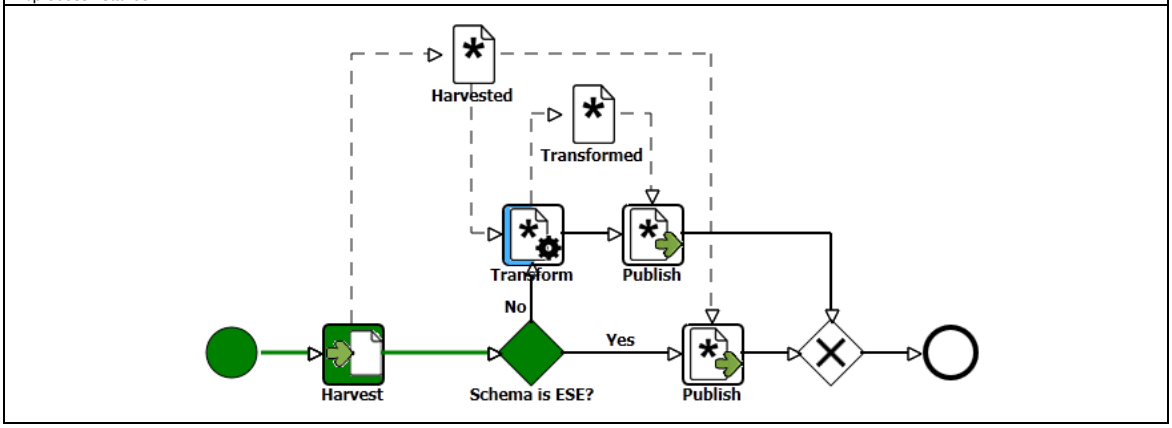
```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL" id="definitions_id_1"
    targetNamespace="http://www.omg.org/bpmn20"
    typeLanguage="http://www.w3.org/2001/XMLSchema"
    expressionLanguage="http://www.w3.org/1999/XPath"
    xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
    xs:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL BPMN20.xsd" xmlns:tns="http://www.jboss.org/drools">
  <! --This <process> part represents the BPMN 2.0 process definition, used within the jBPM to run the process -->
  <process xmlns:param="http://www.omg.org/spec/BPMN/20100524/MODEL" id="process_id_1" processType="Public">
    <! – These properties are used for input and output exchange between tasks in the process -->
    <property id="transformRSTaskParams_0"/>
    <property id="transformRS_result_0"/>
    <property id="publishRSTaskParams_1"/>
    <property id="publishRS_result_1"/>
    <property id="publishRSTaskParams_0"/>
    <property id="publishRS_result_0"/>
    <property id="harParams_0"/>
    <property id="result_0"/>
    <startEvent name="" id="start_id_1"/>
    <! –Each task has the structure described in Section 5.2.2.1 of this document -->
    <task id="harvestTask_id_1" name="Harvest" tns:taskName="IngestWI">
      <ioSpecification>
        <dataInput id="hrparam_0" name="harParams_0"/>
        <dataOutput id="reslt_0" name="HarvestResult"/>
        <inputSet>
          <dataInputRefs>hrparam_0</dataInputRefs>
        </inputSet>
        <outputSet>
          <dataOutputRefs>reslt_0</dataOutputRefs>
        </outputSet>
      </ioSpecification>
      <dataInputAssociation>
        <sourceRef>harParams_0</sourceRef>
        <targetRef>hrparam_0</targetRef>
      </dataInputAssociation>
      <dataOutputAssociation>
        <sourceRef>reslt_0</sourceRef>
        <targetRef>result_0</targetRef>
      </dataOutputAssociation>
    </task>
    <task id="publishTask_id_1" name="Publish" tns:taskName="PublishRSWI">
      <ioSpecification>
        <dataInput id="publishRS_prms_0" name="publishRSTaskParams_0"/>
        <dataOutput id="publishRS_reslt_0" name="PublishRSResult"/>
        <inputSet>
          <dataInputRefs>publishRS_prms_0</dataInputRefs>
        </inputSet>
        <outputSet>
          <dataOutputRefs>publishRS_reslt_0</dataOutputRefs>
        </outputSet>
      </ioSpecification>
      <dataInputAssociation>
        <sourceRef>result_0</sourceRef>
        <targetRef>publishRS_prms_0</targetRef>
      </dataInputAssociation>
      <dataOutputAssociation>
        <sourceRef>publishRS_reslt_0</sourceRef>
        <targetRef>publishRS_result_0</targetRef>
      </dataOutputAssociation>
    </task>
    <task id="publishTask_id_2" name="Publish" tns:taskName="PublishRSWI">
      <! -- Same structure as the previous task-->
    </task>
    <task id="transformTask_id_1" name="Transform" tns:taskName="TransformRSWI">
      <! -- Same structure as the previous tasks, but specified for the transform task-->
    </task>
    <exclusiveGateway gatewayDirection="Diverging" name="Schema is ESE?" id="exclusiveGateway_id_1"/>
    <exclusiveGateway gatewayDirection="Converging" name="" id="exclusiveGateway_id_2"/>
    <endEvent name="" id="endEvent_id_1"/>
    <sequenceFlow sourceRef="start_id_1" targetRef="harvestTask_id_1" name="" id="sequenceFlow_id_1"/>
    <sequenceFlow sourceRef="harvestTask_id_1" targetRef="exclusiveGateway_id_1" name="" id="sequenceFlow_id_2"/>
    <! – Conditions are added to the sequence flows  that follow the diverging exclusive gateway, to decide the workflow of the process-->
    <sequenceFlow sourceRef="exclusiveGateway_id_1" targetRef="transformTask_id_1" name="No" id="sequenceFlow_id_3">
      <conditionExpression xs:type="tFormalExpression">return result_0 == "DIFFERENT_SCHEMA"</conditionExpression>
    </sequenceFlow>
    <sequenceFlow sourceRef="transformTask_id_1" targetRef="publishTask_id_2" name="" id="sequenceFlow_id_4"/>
    <sequenceFlow sourceRef="exclusiveGateway_id_1" targetRef="publishTask_id_1" name="Yes" id="sequenceFlow_id_5">
      <conditionExpression xs:type="tFormalExpression">return result_0 == "SAME_SCHEMA"</conditionExpression>
    </sequenceFlow>
    <sequenceFlow sourceRef="publishTask_id_1" targetRef="exclusiveGateway_id_2" name="" id="sequenceFlow_id_6"/>
    <sequenceFlow sourceRef="exclusiveGateway_id_2" targetRef="endEvent_id_1" name="" id="sequenceFlow_id_7"/>
    <sequenceFlow sourceRef="publishTask_id_2" targetRef="exclusiveGateway_id_2" name="" id="sequenceFlow_id_8"/>
  </process>
  <! – The <processGUI> part stores the required information for visual representation persistency -->
  <processGUI processRef="process_id_1" id="process_id_1_gui">
    <startEventShape eventRef="start_id_1" x="1" y="6" name="" id="start_id_1" state="NOT_PASSED"/>
    <activityShape activityRef="harvestTask_id_1" type="INGEST" x="3" y="6" name="Harvest" id="harvestTask_id_1" state="NOT_PASSED"/>
    <activityShape activityRef="publishTask_id_1" type="PUBLISH_RS" x="9" y="6" name="Publish" id="publishTask_id_1" state="NOT_PASSED"/>
```

```
    <activityShape activityRef="publishTask_id_2" type="PUBLISH_RS" x="8" y="4" name="Publish" id="publishTask_id_2" state="NOT_PASSED"/>
    <activityShape activityRef="transformTask_id_1" type="TRANSFORM_RS" x="6" y="4" name="Transform" id="transformTask_id_1" state="NOT_PASSED"/>
    <entityShape x="5" y="1" name="Harvested" id="entity_id_1" type="RECORD_SET" sourceRef="harvestTask_id_1"
                 targetRef_0="publishTask_id_1" targetRef_1="transformTask_id_1"/>
    <entityShape x="7" y="2" name="Transformed" id="entity_id_2" type="RECORD_SET" sourceRef="transformTask_id_1" targetRef_0="publishTask_id_2"/>
    <gatewayShape gatewayRef="exclusiveGateway_id_1" x="6" y="6" name="Schema is ESE?" id="exclusiveGateway_id_1"
                 gatewayDirection="Diverging" type="EXCLUSIVE" decisionType="CHECK_SCHEMA" state="NOT_PASSED"/>
    <gatewayShape gatewayRef="exclusiveGateway_id_2" x="11" y="6" name="" id="exclusiveGateway_id_2"
                 gatewayDirection="Converging" type="EXCLUSIVE" decisionType="CHECK_SCHEMA" state="NOT_PASSED"/>
    <endEventShape eventRef="endEvent_id_1" x="13" y="6" name="" id="endEvent_id_1" state="NOT_PASSED"/>
    <! – The sourceGateIndex and targetGateIndex attributes refer to the visual docking gate of the sequence flow in each component -->
    <sequenceFlowConnector sequenceFlowRef="sequenceFlow_id_1" targetRef="harvestTask_id_1" sourceRef="start_id_1"
                 sourceGateIndex="2" targetGateIndex="0" name="" id="sequenceFlow_id_1" state="NOT_PASSED" type="FLOW"/>
    <sequenceFlowConnector sequenceFlowRef="sequenceFlow_id_2" targetRef="exclusiveGateway_id_1" sourceRef="harvestTask_id_1"
                 sourceGateIndex="2" targetGateIndex="0" name="" id="sequenceFlow_id_2" state="NOT_PASSED" type="FLOW"/>
    <sequenceFlowConnector sequenceFlowRef="sequenceFlow_id_3" targetRef="transformTask_id_1" sourceRef="exclusiveGateway_id_1"
                 sourceGateIndex="1" targetGateIndex="3" name="No" id="sequenceFlow_id_3" state="NOT_PASSED" type="FLOW"/>
                 <! – Sequence flows of type EXCHANGE are only represented visually and not on the process execution, because they help to determine the data
                 exchange flow and not the workflow of the process-->
    <sequenceFlowConnector targetRef="entity_id_2" sourceRef="transformTask_id_1" sourceGateIndex="1" targetGateIndex="0" name=""
                 id="entityFlow_id_4" state="NOT_PASSED" type="EXCHANGE"/>
    <sequenceFlowConnector sequenceFlowRef="sequenceFlow_id_4" targetRef="publishTask_id_2" sourceRef="transformTask_id_1"
                 sourceGateIndex="2" targetGateIndex="0" name="" id="sequenceFlow_id_4" state="NOT_PASSED" type="FLOW"/>
    <sequenceFlowConnector targetRef="publishTask_id_2" sourceRef="entity_id_2" sourceGateIndex="2"
                 targetGateIndex="1" name="" id="sequence_flow_id_10" state="NOT_PASSED" type="EXCHANGE"/>
    <sequenceFlowConnector sequenceFlowRef="sequenceFlow_id_5" targetRef="publishTask_id_1" sourceRef="exclusiveGateway_id_1"
                 sourceGateIndex="2" targetGateIndex="0" name="Yes" id="sequenceFlow_id_5" state="NOT_PASSED" type="FLOW"/>
    <sequenceFlowConnector sequenceFlowRef="sequenceFlow_id_6" targetRef="exclusiveGateway_id_2" sourceRef="publishTask_id_1"
                 sourceGateIndex="2" targetGateIndex="0" name="" id="sequenceFlow_id_6" state="NOT_PASSED" type="FLOW"/>
    <sequenceFlowConnector sequenceFlowRef="sequenceFlow_id_7" targetRef="endEvent_id_1" sourceRef="exclusiveGateway_id_2" sourceGateIndex="2"
                 targetGateIndex="0" name="" id="sequenceFlow_id_7" state="NOT_PASSED" type="FLOW"/>
    <sequenceFlowConnector sequenceFlowRef="sequenceFlow_id_8" targetRef="exclusiveGateway_id_2" sourceRef="publishTask_id_2"
                 sourceGateIndex="2" targetGateIndex="1" name="" id="sequenceFlow_id_8" state="NOT_PASSED" type="FLOW"/>
    <sequenceFlowConnector targetRef="publishTask_id_1" sourceRef="entity_id_1" sourceGateIndex="2" targetGateIndex="1" name="" id="entityFlow_id_1"
                 state="NOT_PASSED" type="EXCHANGE"/>
    <sequenceFlowConnector targetRef="entity_id_1" sourceRef="harvestTask_id_1" sourceGateIndex="1" targetGateIndex="0" name="" id="entityFlow_id_2"
                 state="NOT_PASSED" type="EXCHANGE"/>
    <sequenceFlowConnector targetRef="transformTask_id_1" sourceRef="entity_id_1" sourceGateIndex="3" targetGateIndex="0" name="" id="entityFlow_id_3"
                 state="NOT_PASSED" type="EXCHANGE"/>
  </processGUI>
  <! – The <parameters> section stores the input information required for each task-->
  <parameters>
    <! – In the next Harvest task, a full harvest is performed to the data source of ID "UniversidadeAberta"-->
    <parameter type="HARVEST" taskID="harvestTask_id_1" paramID="harParams_0" resultID="result_0" sendMail="false"
                 dsID="UniversidadeAberta" fullIngest="true"/>
    <parameter type="PUBLISH_RS" taskID="publishTask_id_1" paramID="publishRSTaskParams_0" resultID="publishRS_result_0" sendMail="true"/>
    <parameter type="PUBLISH_RS" taskID="publishTask_id_2" paramID="publishRSTaskParams_1" resultID="publishRS_result_1" sendMail="true"/>
    <parameter type="TRANSFORM_RS" taskID="transformTask_id_1" paramID="transformRSTaskParams_0"
                 resultID="transformRS_result_0" sendMail="true"/>
  </parameters>
</definitions>
```



C. Example of a simple harvest process instance

```
<! – A process instance is used for process runtime monitoring, and uses it's associated process definition for execution -->
<processInstance xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
        xmlns:param="http://www.omg.org/spec/BPMN/20100524/MODEL" state="RUNNING">
  <processGUI id="process_rpm_gui_id_1" processRef="process_rpm_id_1">
        <! – Every time a process reaches a certain component during the workflow, it updates that component's <state>-->
          <startEventShape eventRef="start_id_1" id="start_shape_1" name="" state="PASSED" x="1" y="6"/>
          <activityShape activityRef="harvestTask_id_1" id="harvest_shape_1" name="Harvest" state="PASSED" type="INGEST" x="3" y="6"/>
          <activityShape activityRef="publishTask_id_1" id="publish_shape_1" name="Publish" state="NOT_PASSED" type="PUBLISH_RS" x="9" y="6"/>
          <activityShape activityRef="publishTask_id_2" id="publish_shape_2" name="Publish" state="NOT_PASSED" type="PUBLISH_RS" x="8" y="4"/>
          <activityShape activityRef="transformTask_id_1" id="transform_shape_1" name="Transform" state="WORKING"
              type="TRANSFORM_RS" x="6" y="4"/>
          <entityShape id="entity_id_1" name="Harvested" sourceRef="harvestTask_id_1" targetRef_0="publishTask_id_1"
              targetRef_1="transformTask_id_1" type="RECORD_SET" x="5" y="1"/>
          <entityShape id="entity_id_2" name="Transformed" sourceRef="transformTask_id_1" targetRef_0="publishTask_id_2"
              type="RECORD_SET" x="7" y="2"/>
          <gatewayShape decisionType="CHECK_SCHEMA" gatewayDirection="Diverging" gatewayRef="exclusiveGateway_id_1"
              id="exclusiveGateway_shape_1" name="Schema is ESE?" state="PASSED" type="EXCLUSIVE" x="6" y="6"/>
          <gatewayShape decisionType="CHECK_SCHEMA" gatewayDirection="Converging" gatewayRef="exclusiveGateway_id_2"
              id="exclusiveGateway_shape_2" name="" state="NOT_PASSED" type="EXCLUSIVE" x="11" y="6"/>
          <endEventShape eventRef="endEvent_id_1" id="endEvent_shape_1" name="" state="NOT_PASSED" x="13" y="6"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_1" sourceGateIndex="2" sourceRef="start_id_1"
              state="PASSED" targetGateIndex="0" targetRef="harvestTask_id_1" type="FLOW"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_2" sourceGateIndex="2"
              sourceRef="harvestTask_id_1" state="PASSED" targetGateIndex="0" targetRef="exclusiveGateway_id_1" type="FLOW"/>
          <sequenceFlowConnector name="No" sequenceFlowRef="sequenceFlow_id_3" sourceGateIndex="1"
              sourceRef="exclusiveGateway_id_1" state="NOT_PASSED" targetGateIndex="3" targetRef="transformTask_id_1" type="FLOW"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_4" sourceGateIndex="1"
              sourceRef="transformTask_id_1" state="NOT_PASSED" targetGateIndex="0" targetRef="entity_id_2" type="EXCHANGE"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_5" sourceGateIndex="2"
              sourceRef="transformTask_id_1" state="NOT_PASSED" targetGateIndex="0" targetRef="publishTask_id_2" type="FLOW"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_6" sourceGateIndex="2"
              sourceRef="entity_id_2" state="NOT_PASSED" targetGateIndex="1" targetRef="publishTask_id_2" type="EXCHANGE"/>
          <sequenceFlowConnector name="Yes" sequenceFlowRef="sequenceFlow_id_7" sourceGateIndex="2"
              sourceRef="exclusiveGateway_id_1" state="NOT_PASSED" targetGateIndex="0" targetRef="publishTask_id_1" type="FLOW"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_8" sourceGateIndex="2"
              sourceRef="publishTask_id_1" state="NOT_PASSED" targetGateIndex="0" targetRef="exclusiveGateway_id_2" type="FLOW"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_9" sourceGateIndex="2"
              sourceRef="exclusiveGateway_id_2" state="NOT_PASSED" targetGateIndex="0" targetRef="endEvent_id_1" type="FLOW"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_10" sourceGateIndex="2"
              sourceRef="publishTask_id_2" state="NOT_PASSED" targetGateIndex="1" targetRef="exclusiveGateway_id_2" type="FLOW"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_11" sourceGateIndex="2"
              sourceRef="entity_id_1" state="NOT_PASSED" targetGateIndex="1" targetRef="publishTask_id_1" type="EXCHANGE"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_12"
              sourceGateIndex="1" sourceRef="harvestTask_id_1" state="NOT_PASSED" targetGateIndex="0" targetRef="entity_id_1" type="EXCHANGE"/>
          <sequenceFlowConnector name="" sequenceFlowRef="sequenceFlow_id_13" sourceGateIndex="3"
              sourceRef="entity_id_1" state="NOT_PASSED" targetGateIndex="0" targetRef="transformTask_id_1" type="EXCHANGE"/>
  </processGUI>
  <parameters>
        <parameter dsID="UniversidadeAberta" fullIngest="true" paramID="harParams_0" resultID="result_0"
              sendMail="true" taskID="harvestTask_id_1" type="HARVEST"/>
        <parameter paramID="publishRSTaskParams_0" resultID="publishRS_result_0" sendMail="true" taskID="publishTask_id_1" type="PUBLISH_RS"/>
        <parameter paramID="publishRSTaskParams_1" resultID="publishRS_result_1" sendMail="true" taskID="publishTask_id_2" type="PUBLISH_RS"/>
        <parameter paramID="transformRSTaskParams_0" resultID="transformRS_result_0"
              sendMail="true" taskID="transformTask_id_1" type="TRANSFORM_RS"/>
  </parameters>
</processInstance>
```



D. Example of a simple harvest process instance log file

```
<logMessages>
  <! – Messages can be sent to the process instance's log asynchronously, to be shown while monitoring a single process-->
  <message taskID="START" messageType="STRING" message="Process Started" startedDate="2011-10-02 15:49:50"/>
  <message taskID="task_id_1" messageType="STRING" message="Harvest on UniversidadeAberta started"
        startedDate="2011-10-02 15:50:04" spentTime=""/>
  <! – Messages can have a<spentTime> which represents the time the described action on the message took to execute -->
  <message taskID="task_id_1" messageType="STRING" message="Harvest Ended with records = 1.040"
        startedDate="2011-10-02 15:51:57" spentTime="00:01:53"/>
  <message taskID="task_id_2" messageType="STRING" message="Schema is NOT of type ESE" startedDate="2011-10-02 15:51:57"/>
  <message taskID="task_id_2" messageType="STRING" message="Transforming record set to ESE schema" startedDate="2011-10-02 15:51:58"/>
  <message taskID="task_id_2" messageType="STRING" message="Record set successfully transformed" startedDate="2011-10-02 15:52:01"
        spentTime="00:00:04"/>
  <message taskID="task_id_3" messageType="STRING" message="Publishing data to storage system" startedDate="2011-10-02 15:52:01"/>
  <message taskID="task_id_3" messageType="STRING" message="Data being published to jdbc:postgresql://http://bd2.inesc-id.pt:8080/repoxdb_new"
        startedDate="2011-10-02 15:52:02"/>
  <message taskID="task_id_3" messageType="STRING" message="Data Published" startedDate="2011-10-02 15:52:05" spentTime="00:00:04"/>
  <message taskID="END_1" messageType="STRING" message="Process Ended" startedDate="2011-10-02 15:52:06" spentTime="00:02:54"/>
</logMessages>
```

Table shown during the single process instance view, based on the log file.

Started Date	Spent Time	Message
2011-10-02 15:49:50		Process Started
2011-10-02 15:50:04		Harvest on UniversidadeAberta started
2011-10-02 15:51:57	00:01:53	Harvest Ended with records = 1.040
2011-10-02 15:51:57		Schema is NOT of type ESE
2011-10-02 15:51:58		Transforming record set to ESE schema
2011-10-02 15:52:01	00:00:04	Record set successfully transformed
2011-10-02 15:52:01		Publishing data to storage system
2011-10-02 15:52:02		Data being published to jdbc:postgresql://
2011-10-02 15:52:05	00:00:04	Data Published
2011-10-02 15:52:06	00:02:54	Process Ended

E. Example of a system's process definitions file

```
<processes>
        <! – This file is used to show all process definitions, including their title, visual SVG representation and creation date -->
        <process id="process_rpm_190595f0-92a4-406a-886f-f3539000c3f6" title="finalTEST" createdDate="2011-06-17 11:45:30"/>
        <process id="process_rpm_5245e543-fcf1-4b9d-bc2e-a771553116d0" title="big1nnn" createdDate="2011-06-17 13:43:39"/>
        <process id="process_rpm_c8bc1b9a-dcf3-4608-bd32-266649b4e2ef" title="simple process" createdDate="2011-08-25 15:34:49"/>
        <process id="process_rpm_5a786974-c064-49a9-8425-e85be1d666e3" title="test send data" createdDate="2011-08-25 15:35:16"/>
        <process id="process_rpm_a087a8e3-55f1-4842-a937-3282270e59a7" title="parallel ingestion test" createdDate="2011-08-25 15:35:36"/>
        <process id="process_rpm_7d7c1657-8b50-4456-a0a0-1711a4dbbef4" title="simple harvest" createdDate="2011-08-25 15:35:51"/>
        <process id="process_rpm_87aa3a36-b271-4118-a43b-359cbbe8baa1" title="test complex process" createdDate="2011-08-25 15:37:00"/>
        <process id="process_rpm_b0aaba83-ab86-4df5-8a76-c59e698b4eae" title="test package process" createdDate="2011-08-25 15:37:46"/>
</processes>
```

F. Example of a system's process instances file

```
<processes>
        <! – This file is used to show all process instances, including their execution state, comments and visual runtime representation -->
        <process id="rpm_4e3d3d25-10f8-4952-9245-dfa4099ec7f0_inst_testNEWas"
              processDefinitionID="process_rpm_b0aaba83-ab86-4df5-8a76-c59e698b4eae"
              processDefinitionTitle="test simple harvest" state="COMPLETED" startedDate="2011-06-22 17:57:33" comment=""/>
        <process id="rpm_d4c0248a-cc76-4695-b9a2-7fcb23eba248_inst_testNEWas"
              processDefinitionID="process_rpm_b0aaba83-ab86-4df5-8a76-c59e698b4eae"
              processDefinitionTitle="testNEWas" state="COMPLETED" startedDate="2011-06-22 18:00:12" comment="test comment"/>
        <process id="rpm_4dfe3a76-d1b4-49c2-9afb-7dc72603275a_inst_testcomplex"
              processDefinitionID="process_rpm_87aa3a36-b271-4118-a43b-359cbbe8baa1"
              processDefinitionTitle="testcomplex" state="COMPLETED" startedDate="2011-06-24 12:02:49" comment=""/>
        <process id="rpm_f4cc385e-65ea-4488-9dbe-8c007285370b_inst_big1nnn"
              processDefinitionID="process_rpm_5245e543-fcf1-4b9d-bc2e-a771553116d0"
              processDefinitionTitle="big1nnn" state="RUNNING" startedDate="2011-06-24 15:00:06" comment=""/>
        <process id="rpm_c3dd8e0f-a685-461d-a437-61e2292ecd62_inst_ssssimplee"
              processDefinitionID="process_rpm_c8bc1b9a-dcf3-4608-bd32-266649b4e2ef"
              processDefinitionTitle="simplee" state="RUNNING" startedDate="2011-06-24 15:28:32" comment=""/>
</processes>
```

G. REPOX 2.0 new web interface

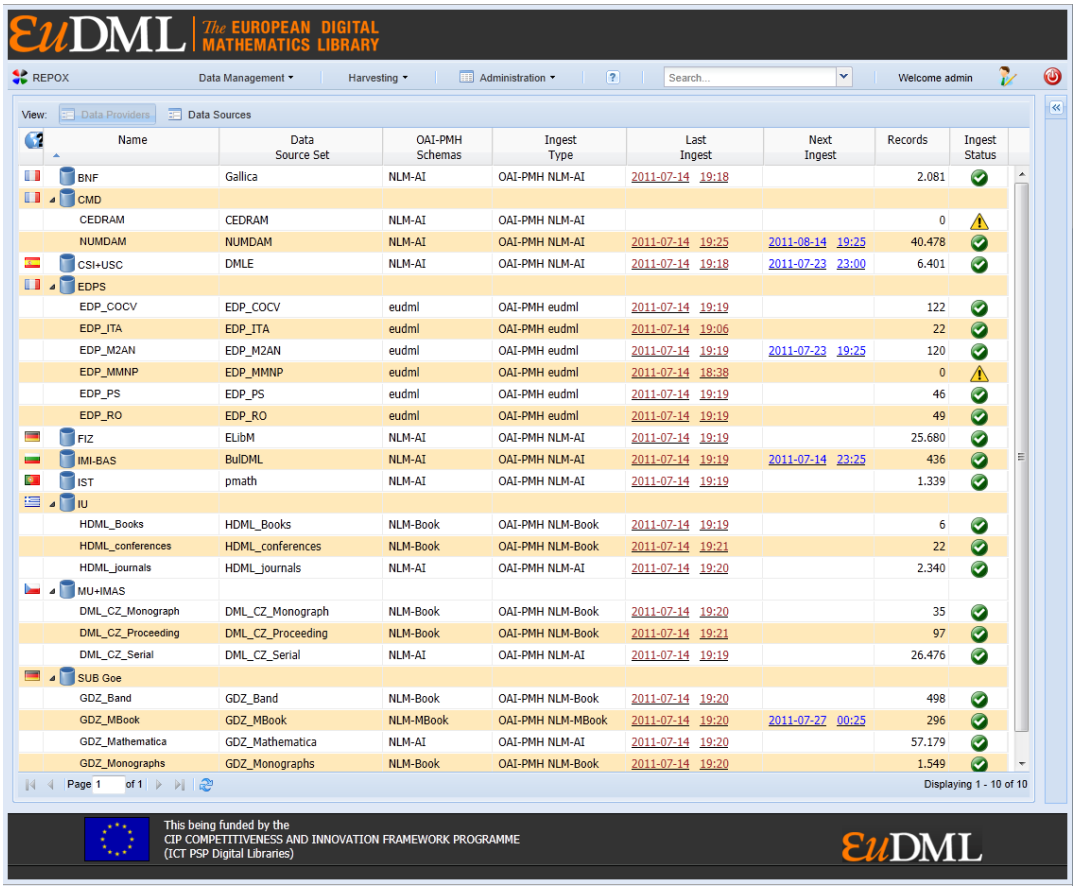


Figure 30. REPOX 2.0 new web interface for the EuDML scenario.

The previous figure shows the new REPOX 2.0 web interface customized to the EuDML scenario. On the previous versions of REPOX, the main interface used a grid to display data providers and their data sources, which lead us, due to its familiarity by users, to use same approach but now migrated from Stripes[79] to the GWT framework. We also added new features like a user management system for REPOX administrators and links in the *Last* and *Next Ingest* dates shown in the previous figure that are connected to a embedded calendar that displays scheduled and old harvests (with a resume and associated log file). GWT also allowed us to group the data sources of a data provider under it through a tree-like approach. The development of this interface allowed a more thorough comprehension of the main concepts in REPOX, which is used for digital libraries' interoperability management.

[79] Stripes Framework for Web development - http://www.stripesframework.org/

H. Acronym Glossary

Acronym	Definition
BPD	Business Process Diagram
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMI	Business Process Management Initiative
BPMN	Business Process Modeling Notation
CSS	Cascading Style Sheets
DC	Dublin Core
DHTML	Dynamic HyperText Markup Language
DOM	Document Object Model
DSL	Domain Specific Language
DSVL	Domain Specific Visual Language
DTD	Document Type Definition
EAD	Encoded Archival Description
ECM	Enterprise Content Management
ESE	Europeana Semantic Elements
ETL	Extract, Transform, Load
EuDML	European Digital Mathematics Library
FTP	File Transfer Protocol
GWT	Google Web Toolkit
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
jBPM	Java Business Process Management
LIDO	Lightweight Information Describing Objects
MARC	Machine Readable Cataloging standards
NSDL	National Science Digital Library
OAI-PMH	Open Archives Initiative Protocol for Metadata Harvesting
UI	User Interface
UML	Unified Modeling Language
UNIMARC	Universal Machine Readable Cataloging
VPL	Visual Programming Language
VPML	Visual Process Modeling Language
WS-BPEL	Web Services Business Process Execution Language
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations
ZUML	ZK User Interface Markup Language