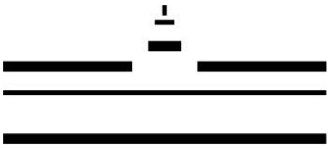


Title:

A Prototype for Semantic Process Intelligence

Master Thesis

at the Chair for Information Systems and Information Management
Department of Information Systems
University of Muenster

in cooperation with the Center for Business Process Innovation
Howe School of Technology Management
Stevens Institute of Technology

Supervisor:	Prof. Dr. Michael zur Muehlen Stevens Institute of Technology
	Prof. Dr. Jörg Becker University of Muenster
Advisor:	Dr. Armin Stein University of Muenster
Presented by:	David Overfeld Forellengrund 15 49479 Ibbenbüren david.overfeld@uni-muenster.de
Date of Submission:	2014-08-22

If you don't understand what your software engineers are talking about, perhaps it's because they are using a vocabulary they invented for the problem they are solving.

– Moran (2013)

Content

Figures

Tables

Queries

Abbreviations

API	Application Programming Interface
BPI	Business Process Intelligence
BI	Business Intelligence
BPM	Business Process Management
BPMN	Business Process Model and Notation
BPMS	Business Process Management System
CEP	Complex Event Processing
CRM	Customer Relationship Management
ERP	Enterprise Resource Planning
IDE	Integrated Development Environment
RDF	Resource Description Framework
SLA	Service Level Agreement
SPARQL	SPARQL Protocol and RDF Query Language
SCM	Supply Chain Management
OWL	Web Ontology Language
W3C	World Wide Web Consortium
WFM	Workflow Management
WFMS	Workflow Management System
XPDL	XML Process Definition Language

1 Introduction

1.1 Problem Statement

Business Process Intelligence (BPI) provides process participants, decision makers and related stakeholders with insight about the effectiveness and efficiency of organizational processes (Grigori et al. 2004; zur Muehlen and Shapiro 2010). Most process analyses are based on the aggregation, correlation and evaluation of events that occur during the execution of a process, i.e. state changes of objects within the context of a business process. The relevant events are generated from different components within the IT infrastructure such as business process management (BPM), enterprise resource planning (ERP) or customer relationship management (CRM) systems (van der Aalst et al. 2007; zur Muehlen and Shapiro 2010) and are usually saved in relational databases (Martinez-Cruz et al. 2012). To be able to create analyses based on all relevant events, data integration has to be performed dealing with the problems arising from structural as well as semantic heterogeneity of different data sources. Depending on the number of different data sources, data integration can be complex and costly (van der Aalst et al. 2007; Bergamaschi et al. 2011). The problem of heterogeneity is mostly due to the use of relational databases that are efficient for storing and retrieving data in and from a single data source but lack a sufficient machine readable semantic meaning of the stored data which makes it difficult to integrate different data sources (Martinez-Cruz et al. 2012).

Existing xml-based e-Business standards like XPDL, BPAF or ebXML seek to provide unambiguous specifications for error-free exchange of process event data between different systems. These standards are, however, syntax-based and do not guarantee semantic interoperability between different partners (Heravi et al. 2010). Furthermore, those standards are not reflected in the internal databases of the event generating systems and standard-conform documents have to be generated from the existing data (Genrich et al. 2008; Janiesch et al. 2011).

Semantic Web technologies were developed with the idea in mind to integrate data from arbitrary (web-) sources. Although originally developed for the web, they are generally suited for every scenario of data integration (Martinez-Cruz et al. 2012). Despite the ongoing discussion about Semantic Web technologies and their application possibilities in academia, the industry still hesitates to make use of the concepts. This is, on the one hand, due to the Semantic Web's inherent complexity but, on the other hand, due to a lack of good examples and prototypes showing the potential in a real-life context.

1.2 Purpose and Research Questions

In order to address the aforementioned data integration problems, this thesis aims at developing a semantically unambiguous standard for process event data exchange based on ontologies. An ontology-based event format has the advantage that event data from arbitrary sources (when saved in this format) can be combined without the necessity for additional data integration. This enables a faster and more reliable event analysis. Furthermore, ontological event data can directly be queried using the Semantic Web query language SPARQL making a distinction between exchange and analysis format obsolete. Finally, reasoning over events from different systems and asking cross-system questions are possible. This leads to the following research question:

RQ 1: How can an ontology for business process event data be built?

To build an ontology, an appropriate methodology for the development process has to be identified and requirements for the ontology have to be collected. For the latter, this thesis aims at reusing existing knowledge about process and event exchange formats by extracting components that can be represented in an ontological form.

An event ontology only provides the basis for semantic process intelligence. Appropriate event import and analysis functions are necessary in order to turn event data into decision relevant information. This leads to the next research question:

RQ 2: How can a prototype for semantic process intelligence be built?

To build a prototype, requirements have to be defined first. Then, based on the developed ontology, appropriate SPARQL queries for extracting desired information such as time metrics for activities and processes have to be developed. Due to the complexity of the Semantic Web's standards, the use of existing Semantic Web application frameworks is reasonable. Therefore, a review and selection of an appropriate framework is necessary. Eventually, an appropriate application architecture has to be designed that is flexible enough to be used with arbitrary event generating systems.

1.3 Thesis Structure

The thesis is structured as follows: Chapter 2 discusses the research method of the thesis where the design science approach was chosen as the appropriate method. Further, the individual sections of the thesis are related to the stages of the approach.

Chapter 3 covers the necessary foundations of the thesis and introduces important terms and concepts used throughout this thesis. Section 3.1 covers workflow management and business process management and focuses on the historical development as well as the differences and similarities of both disciplines. Section 3.2 serves as an introduction to the area of business process intelligence whereby common application areas as well as challenges are presented. Section 3.3 discusses the differences between semantic and non-semantic technology and highlights both its advantages and disadvantages. Section 3.4 covers the Semantic Web and its standards used for the prototype development while section 3.5 summarizes related work.

Chapter 4 deals with the ontology development. Therefore, the development process including the chosen methodology, ontology language and ontology development tool are presented first (section 4.1). Subsequently, the ontology requirements are specified (section 4.2), important ontology components are conceptualized (section 4.3) and eventually implemented (section 4.4). Chapter 5 demonstrates the ontology by using synthetic process event data translated to its ontological representation and applying developed SPARQL queries answering at set of defined test questions.

Chapter 6 presents the prototype development. It starts with the definition of functional requirements that should be fulfilled by the prototype (section 6.1). The following section (6.2) summarizes the results of the conceptualization phase presenting the chosen semantic application framework, web development framework and general prototype architecture. In addition, the BPMS chosen to evaluate the prototype with real-life event data are introduced. Section 6.3 explains important components of the implemented prototype. Chapter 7 demonstrates and evaluates the prototype using synthetic as well as real-life event data.

Chapter 8 discusses the results, summarizes important lessons learned during the ontology and prototype development and concludes the thesis with implications for future research.

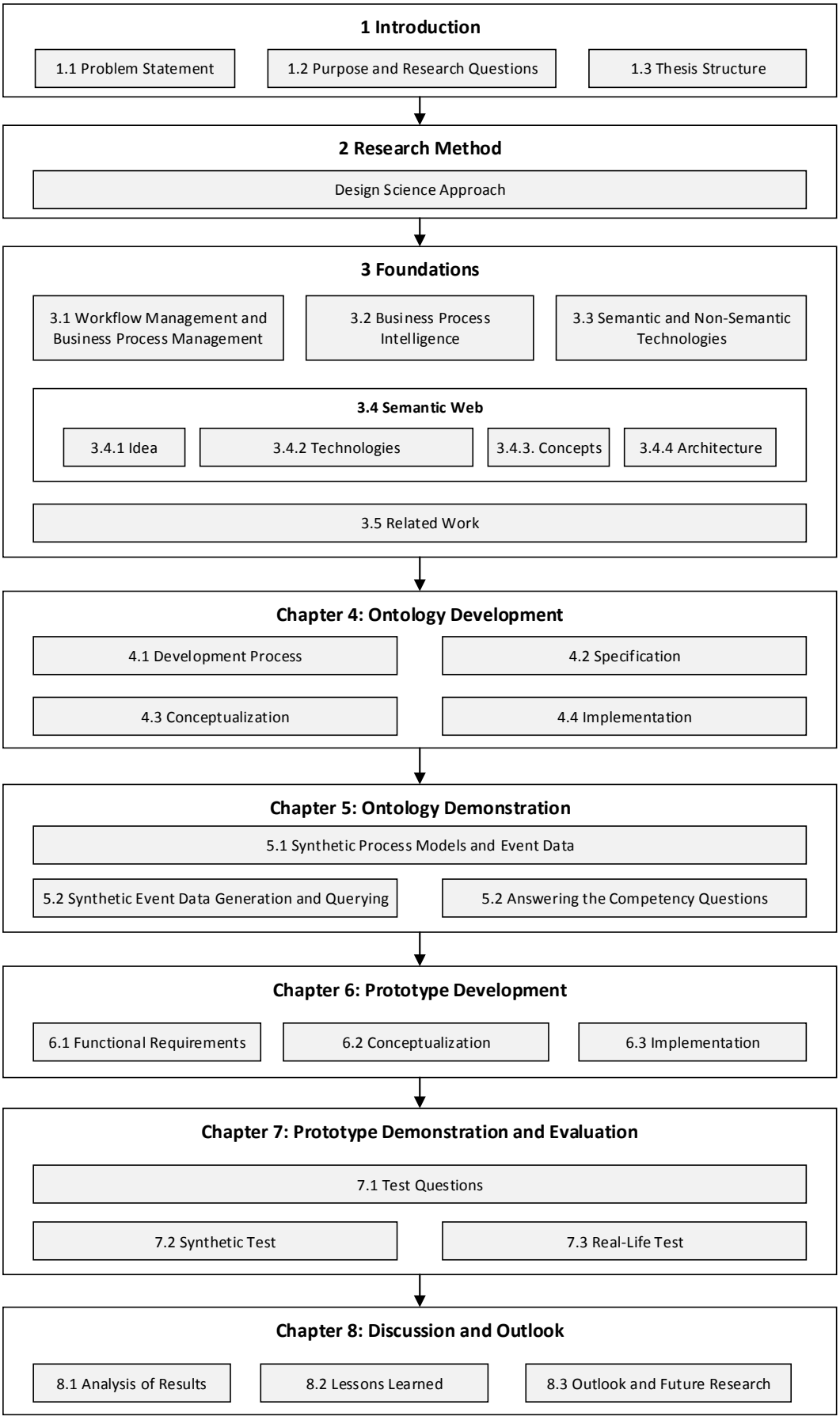


Figure 1-1 Structure of the Thesis

2 Research Method

This thesis follows the design science approach. Design science can be understood as the scientific study of design (Gregory 1966). In specific, design science attempts to create artifacts that serve human purposes (Simon 1996). Due to its application in various research areas and its adaptable nature, different definitions of the design science approach exist. In general, the six different steps depicted in Figure 2-1 can be distinguished (Peffers et al. 2007). Even though the process is structured in a nominally sequential order starting with step one, PEFFERS et al. (2007) point out that the exact entry point can vary: A problem-centered approach is the basis of the nominal sequence, starting with activity one. Researchers might proceed in this sequence if the idea for the research resulted from an observation of the problem or from suggested future research in a paper from a prior project. An objective-centered solution starting with activity two, could be triggered by an industry or research need that can be addressed by developing an artifact. As described in the problem statement (section 1.1), there is a lack of integration between different event generating systems making the analysis of those events difficult. This circumstance motivates a problem-centered design science approach. In the following, each individual step of the design science approach is described and associated with the deliverables of this thesis.

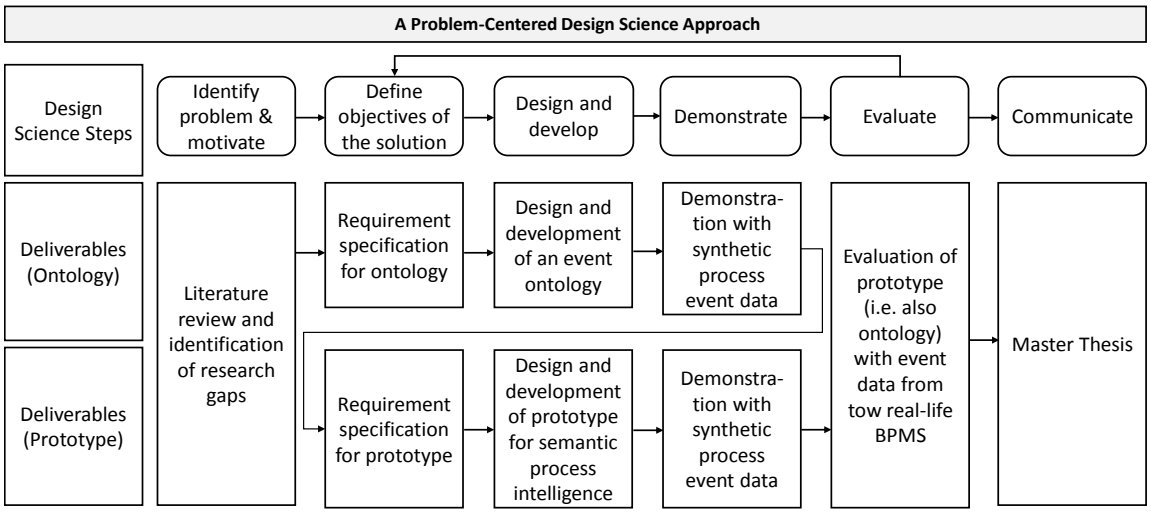


Figure 2-1 Applied Design Science Approach

In order to extend the problem identification and motivate the thesis (step one), Workflow Management, Business Process Management and Business Process Intelligence (BPI) are presented whereby current challenges of BPI are highlighted. Besides, the principles of the Semantic Web and its underlying technologies are presented, in order to provide the

necessary technical background for the developed ontology and prototype as well as to emphasize their relevance in the area of Business Process Intelligence.

According to PEFFERS et al. (2007), the following four steps (two to five) are conducted iteratively in order to incrementally improve the prototype within each round. The sequence of the four steps allows for the design and implementation of basic functionalities in a prototype first. In the subsequent iterations, the prototype is redefined step-by-step until all requirements are met and the prototype has reached a sufficient level of maturity (Gregor and Hevner 2013). This principle was followed throughout this thesis. As depicted in Figure 2-1, steps two, three and four were repeated twice in each iteration. First, for the ontology and then for the prototype. This was important for tracing back errors since it made sure that an error occurring in the prototype was not due to problems in the underlying ontology but due to the prototype itself.

In order to define objectives for the ontology and prototype, the area of business process intelligence (see section 3.2) was analyzed and a set of competency questions was derived that the ontology respectively the prototype should be able to answer (see section 4.2).

For the design and development of an event ontology, existing event as well as workflow exchange formats were analyzed. In addition, existing literature originating from the area of event driven business process management (edBPM) on requirements for event exchange formats was studied (see especially section 4.3). The result was a set of general requirements that could be translated into an event ontology. Based on the ontology, a semantic prototype was developed meeting the objectives defined earlier. Due to a lack of comparable prototypes in literature and practice, the presented prototype could not be based on existing knowledge and is the first of its kind.

To be able to demonstrate the general functionality of the ontology, a SPARQL-endpoint (see section 5.2) was set up that could execute SPARQL queries against a set of generated test data (synthetic event data). The same synthetic event data was used to demonstrate the prototype's functionality. To evaluate the readiness for real-life event data, two open source BPMS were chosen (Stardust and Camunda BPMS). For each system, the respective event model was analyzed and an import transferring the event data from the internal representation to the ontological representation was developed. The analysis functions provided by the prototype were then used to evaluate it against a set of defined test questions.

The communication of the results represents the last step of the design science approach and comprises the finalization of this thesis.

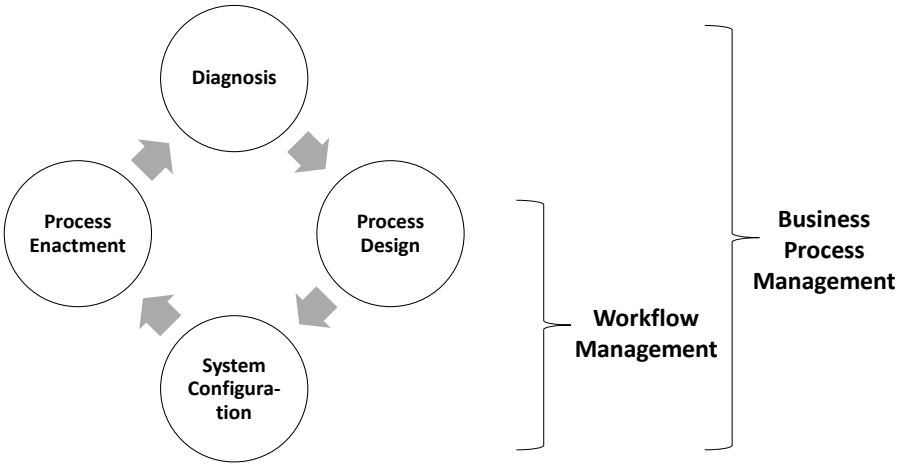
3 Foundations

The following sections introduce all relevant terms and topics used throughout this thesis. The first section deals with workflow management and process management (section 3.1) while the second (3.2) introduces business process intelligence whereby current challenges are highlighted. Sections 3.3 and 3.4 cover the most important concepts and standards of the Semantic Web that form the basis for the later ontology and prototype development. Section 3.5 summarizes related work.

3.1 Workflow Management and Business Process Management

Workflow Management (WFM) is a term that arose during the 1990s and refers to the automation of business processes (van der Aalst et al. 2003). A process can be defined as a completely closed, time-logical sequence of activities that are required for working on a process-oriented relevant business object (Becker and Kahn 2003). The goal of WFM is hence to make sure that the right activities are executed by the right person at the right time (van der Aalst 1998). In practice, WFM is mostly established through Workflow Management Systems (WFMS) (van der Aalst 1998). The Workflow Management Coalition (WFMC) defines a WFMS as "a system that completely defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participant and, where required, invoke the use of IT tools and applications." A workflow is defined as the "automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" (Lawrence et al. 1997).

The term Business Process Management (BPM) evolved in the 1990s when researchers and practitioners discovered that pure enactment of business processes through automated workflows is too restrictive. Therefore, BPM can be seen as an extension to Workflow Management (WFM) (van der Aalst et al. 2003). The Workflow Management Coalition (WFMC) defines Business Process Management (BPM) as a discipline involving any combination of modeling, automation, execution, control, measurement and optimization of business activity flows, in support of enterprise goals, spanning systems, employees, customers and partners and beyond the enterprise boundaries (WFMC 2014a). As for WFM, there exist systems for BPM called Business Process Management Systems. These systems which can be seen as successors of WFMS can be defined as a generic system that is driven by explicit process designs to enact and manage operational business processes (van der Aalst et al. 2003).



Source: Aalst et al. (2003), p. 5

Figure 3-1 WFM compared to BPM

Figure 3-1 shows the relationship between WFM and BPM using the BPM lifecycle. The BPM lifecycle describes the various phases of operational business processes. In the design phase, processes are (re)designed. In the configuration phase, the designed processes are implemented by configuring a process aware information system. This phase includes the transformation from a process model to a workflow model which can be enacted by the system. This can, depending on the initial process model and chosen modeling languages, include additional modeling effort. After configuration, the enactment phase starts when the workflows are executed using the configured system. In the diagnosis phase, the operational workflows are analyzed to identify problems or to find potential for improvement. The focus of the traditional WFM is the lower half of the BPM cycle (van der Aalst et al. 2003).

Some authors argue that WFM and BPM in practice are used interchangeably (zur Muehlen and Hansmann 2012). In the context of this thesis, however, BPM is considered to have a broader focus than WFM. Also, the notions of process and workflow are often used interchangeably in practice. For this thesis a workflow is (as described in the above definition) the "executable version" of a process model.

3.2 Business Process Intelligence

Business Process Intelligence (BPI) refers to the application of Business Intelligence (BI) techniques to business processes (Grigori et al. 2004). In this context, BI refers to technologies, applications and practices for the collection, integration, analysis and presentation of business information. The purpose of BI is to support better decision making (Soldic-Aleksic and Stankic 2011). The traditional data source for BI is a data-warehouse used for the collection and processing of historical data from different organizational information systems (Casati et al. 2002).

BPI is an emerging area that is quickly gaining interest due to the increasing pressure companies are facing to improve the efficiency of their business processes and to quickly react to market changes. In addition, the need to meet regulatory compliance has further increased the relevance of BPI for companies (Castellanos et al. 2009). BPI, in general, is used to answer the following questions: (1) what has happened in the past, (2) what is happening currently and (3) what might happen in the future (zur Muehlen and Shapiro 2010).

Process-aware information systems such as WFM, ERP, SCM and CRM system are recording business events occurring during process execution in event logs (Dumas et al. 2005). These event logs can contain information such as start and completion times of activities or information about the resources executing the activities. BPI exploits this process information by providing means for analyzing it and giving companies a better understanding of how their processes are executed. Therefore, BPI often triggers process improvement or reengineering efforts (Castellanos et al. 2009).

3.2.1 Application Areas

In literature, different views on the application areas of BPI exist (for instance: van der Aalst 2012; Castellanos et al. 2009; Grigori et al. 2004; Mühlen and Shapiro 2010). In the following, the view provided by CASTELLANOS ET AL. (2009) will be used. The authors subdivide BPI into the areas of Process Analysis, Process Discovery, Process Monitoring and Conformance Checking which will be explained in detail in the following. Figure 3-2 depicts the different application areas in the context of the event collection.

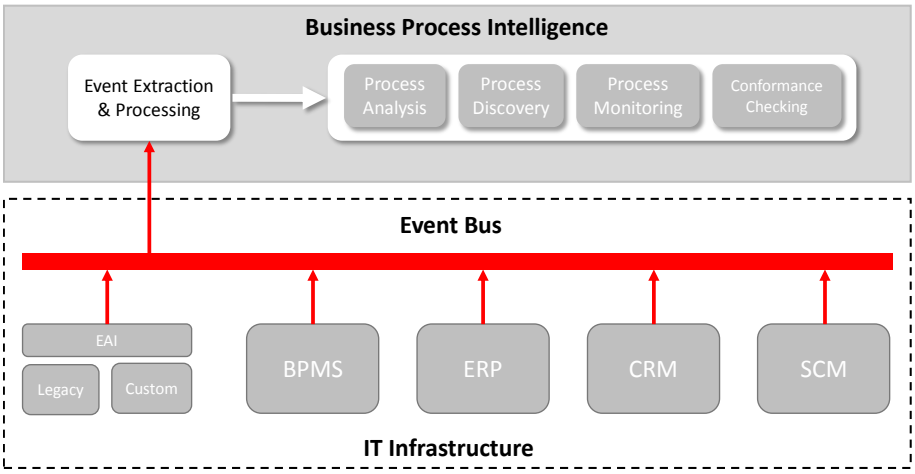


Figure 3-2 Business Process Intelligence in Context

Process Analysis

Process Analysis refers to the analysis of past or current process executions with the aim to create explanatory, decision and prognosis models. Explanatory models shall find correlations between different aspects of workflow executions and performance metrics (e.g. disproportionally high processing costs for certain customers). In addition, analysts are supported in identifying opportunities for process optimization through the identification of causes for malfunctions or bottlenecks (decision models). The aim of prediction models is to predict critical situations and undesired behavior (e.g. exceptional situations or delays on a running process instance that bear the risk of an SLA violation). This enables companies to either prevent or minimize the occurrence of these critical situations by taking corrective measures proactively (Castellanos et al. 2009).

Process Discovery

Process Discovery refers to the analysis of event logs to discover process, control, data, organizational and social structures (van der Aalst et al. 2007). While process analysis primarily focuses on the analysis of business processes in respect to performance metrics, process discovery aims at constructing process models from historical data. Process discovery is also referred to as process mining in literature (van der Aalst 2012).

Process Monitoring

Process monitoring refers to the monitoring of running process instances to inform users about unusual or undesired situations. The aim is to detect problems as early as possible to minimize negative consequences for operational efficiency and customer satisfaction (Grigori et al. 2004). Process dashboards or automatically generated alerts provide the information to react appropriately (Castellanos et al. 2009). Especially businesses that rely on a timely reaction to disruptive events can profit from process monitoring. A typical application area of process monitoring is Supply Chain Management, referred to as Supply Chain Event Management in literature (Otto 2003).

Conformance Checking

The aim of conformance checking is to analyze whether a log conforms to a process model and to identify undesired behavior a-posteriori (Rozinat and van der Aalst 2008). Conformance checking can for instance be used to detect security or compliance violations (e.g. violations of the separation-of-duty principle) in event logs (van der Aalst and de Medeiros 2005).

3.2.2 Challenges

The analysis of business processes with business process intelligence techniques faces several challenges in practice. These challenges can be divided into technical, interpretive and pragmatic challenges (Genrich et al. 2008).

Technical Challenges

Most importantly, business process intelligence has to cope with the heterogeneous systems landscape of large enterprises. While BPI tools can be rather easily used on log data of business processes that are executed by a single workflow management system (van der Aalst et al. 2007), it becomes difficult to integrate log data from different systems (cf. Figure 3-2), especially if diverse technologies are used and different types of events are recorded. Log files have to be extracted and transformed from various formats to one analysis format. Another problem is that not all systems record all relevant events, especially those that of human executed steps, such that they cannot be included in an analysis (Genrich et al. 2008).

Interpretive Challenges

When technical challenges have been overcome, the generated information still has to be interpreted by domain experts. The interpretive challenge stems, among others, from the fact that BPI techniques can only operate on the set of events that is actually recorded for a process. In practice, not all relevant events are actually logged and people may find ways to work around the system (van der Aalst et al. 2007). Even if all data is available, the quality may be too poor to use it directly. Therefore, it is crucial to understand the mindset and motivations of the various agents involved in the execution of the process (Genrich et al. 2008).

Pragmatic Challenges

As soon as technical and interpretive challenges are resolved, pragmatic conclusions can be drawn from the interpretations. The results must be presented in an appropriate manner such that decision makers can derive the right conclusions from them (Corea and Watters 2007). Even if the presentation of the results is appropriate, they cannot be always directly translated into business objectives for staff. The reason is that some objectives enforce undesired behavior of the workforce. This is for example the case when call center agents hang-up on callers in order to improve their number of handled calls. Therefore, the performance objectives concluded from the BPI analysis must be chosen such that they align the behavior of the workforce with performance objectives of the business process (Anderson and Oliver 1987).

3.3 Semantic and Non-Semantic Technology

Computer systems have traditionally been built upon non-semantic data models. These models are designed to satisfy the requirements for a specific system which was developed in response to an identified need or problem. The data model serves to constrain data into a certain structure and application logic is built upon this structure. Meaning comes from the application logic. In fact, for machines, there is no meaning stored with non-semantic data itself. Semantic technology tries to solve this problem by storing the meaning (semantic) with the data itself, thus making it possible for different systems to use the same meaning and the same vocabularies (Moran 2013). This enables information to be combined and used rapidly in ways that are not possible with relational or traditional XML technology (non-semantic technology) (Berners-Lee 1998).

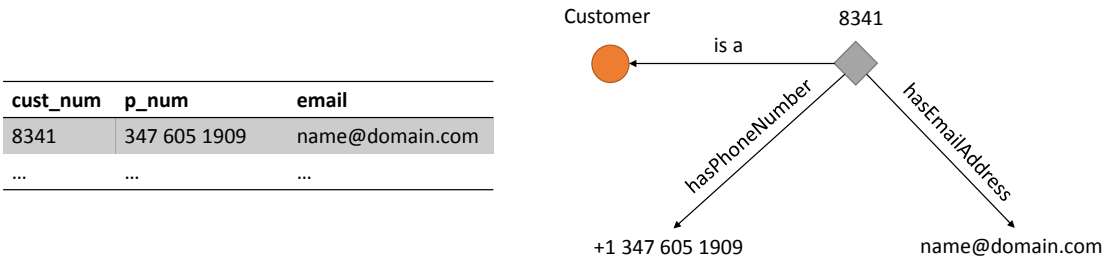


Figure 3-3 Semantic and non-Semantic Representation of Data

Figure 3-3 illustrates the difference between a semantic and a non-semantic representation of data. The table on the left displays data as stored in a relational database. In order to turn this structured data into information, a human must interpret it, such as what *p_num* actually describes. The graph on the right shows the information that a customer has the phone number *347 605 1909* and the email address *name@domain.com*. The meaning of the data on the right is stored with the data itself, thereby making it information. If one wants to improve the understanding of the given information, new information can simply be connected to the given one. There is no necessity for redesigning the data model, changing foreign keys or adding null values. Semantic technology is based on the principle that anyone can say anything about any topic (Allemang and Hendler 2011, p. 323). This triple based principle spans a graph, referred to as graph data (see section 3.4.2).

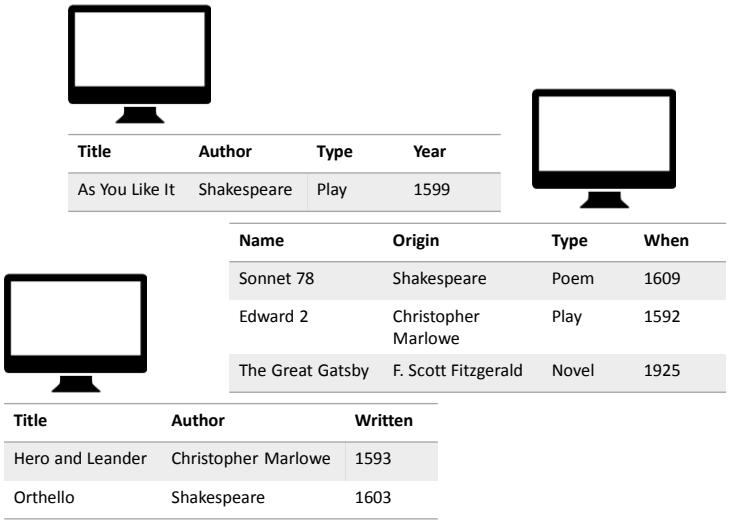
Although semantic data is easier to understand for machines, (traditional) relational data still has an advantage. In cases where lots of similar data has to be stored and retrieved without the need to connect it with other data, relational databases are faster. This is due to the fact that the data is represented in a less machine-understandable, but more efficient form (Martinez-Cruz et al. 2012).

3.4 Semantic Web

In the following, the idea of the Semantic Web will be explained first (section 3.4.1). Afterwards, a basic introduction to Semantic Web technologies will be given (section 3.4.2). Since a complete coverage of all aspects of the described technologies is out of the scope of this thesis, only the main ideas of the respective technologies will be described. Section 3.4.3 summarizes important concepts of the Semantic Web, while section 3.4.4 describes its general architecture.

3.4.1 Idea

At its beginning, the World Wide Web could be seen as a set of web pages offering a collection of web documents. The main goal was to publish content for the audience, i.e. human readers (Allemang and Hendler 2011, p. 7; Yu 2011, p. 9). Web search engines (like Google) had not been developed yet and an integration of information from different sources was not in the focus of web developers. This led to a great diversity of different content representations that are in most cases easy to understand for humans but not for machines. Figure 3-4 depicts a typical case of distributed data on the traditional web where different representations are used for the same kind of data. Three hypothetical web sites *www.shakespeare.com*, *www.literature.com* and *www.playsandpoems.com* are hosting very similar information with a slightly different focus each.



Title	Author	Type	Year
As You Like It	Shakespeare	Play	1599

Name	Origin	Type	When
Sonnet 78	Shakespeare	Poem	1609
Edward 2	Christopher Marlowe	Play	1592
The Great Gatsby	F. Scott Fitzgerald	Novel	1925

Title	Author	Written
Hero and Leander	Christopher Marlowe	1593
Orthello	Shakespeare	1603

Source: Based on Allemang and Hendler (2011), p. 29

Figure 3-4 Distributed Data across the Web

While the first site solely hosts information about plays of Shakespeare, the second is specialized on literature in general. The third hosts information about plays and poems. If one wants to know all plays written by Shakespeare, an automatic integration by machines would be difficult and error prone. The reason is, that all websites are using

different schemas including different column headers. To automatically infer that "Author" and "Origin" actually means the same, would be an easy task for humans but a difficult task for machines. In 2001, the World Wide Web Consortium (W3C) director Tim Berners-Lee formally introduced the idea of the Semantic Web (Berners-Lee et al. 2001), which can be understood as follows: The Semantic Web provides the technologies and standards that are needed to make data published on the web machine-readable. By using semantic web technologies for publishing data, machines can understand the data and automatically accomplish tasks that in the traditional web require human interaction and interpretation (Yu 2011, p. 9). Since then, a dedicated W3C working group has been working on improving, extending and standardizing the idea of the Semantic Web and its underlying technologies as part of the "Semantic Web Activity" (W3C 2014a).

3.4.2 Technologies

To achieve the goal of a machine readable web, the Semantic Web is built upon of a stack of different standards maintained and developed by the World Wide Web Consortium (W3C 2014a) . Before they will be explained in more detail in the following sub-sections, this chapter will give a short overview about the line of reasoning behind the most important standards.

The idea of the Semantic Web is to enable machines to integrate all data available on the internet. To make this vision possible, four things have to be accomplished: First, a general machine-readable format has to exist that can be used to describe everything in the world. This format is the Resource Description Framework (RDF). Second, to avoid misunderstandings and provide means to reason over existing data, a language has to exist that enables the description and publication of allowed vocabulary and the relations between the terms defined in the vocabulary (ontologies). This is accomplished via RDF Schema (RDFS) and the more powerful Web Ontology Language (OWL). Third, there must be means that use the understanding of existing data to verify their correctness and infer additional information (if possible). For this task, so-called reasoners (software applications) are used. Forth, if all data (using RDF) and the underlying semantics (using RDFS and OWL) are published on the web, there has to exist a possibility to easily retrieve and combine those data. Therefore, the SPARQL Protocol and RDF Query Language (SPARQL) was created. The Semantic Web technology stack provides a fourth standard that is used to semantically enrich human readable websites. The so-called Semantic Markup languages or micro formats (e.g. RDFa: Resource Description Framework in Attributes) allow web developers to enrich HTML code with RDF statements directly, describing what certain content means (Yu 2011, p. 87). With this approach, the old and the Semantic Web can be integrated into one web page. Since micro

formats are not part of the prototype, they won't be further explained in the following sections. Table 3-1 summarizes the mentioned standards.

Standard	Short Description
Resource Description Framework (RDF)	Framework for describing the world as triples (anyone can say anything about any topic).
Semantic Markup languages (e.g. RDFa)	Standards for integrating RDF statements into websites.
RDF Schema (RDFS) and Web Ontology Language (OWL)	Standards for defining terms and relationships between terms (ontologies). Used by RDF statements to clarify meaning.
Reasoner	Software components that can infer (i.e. reason) additional knowledge (additional statements) from a given set of statements using rules defined in ontologies.
Protocol and RDF Query Language (SPARQL)	Standard for querying and combining RDF data published on the web.

Table 3-1 Semantic Web Standards

RDF

The Resource Description Framework (RDF) is based upon the idea of making statements about resources in the form of subject-predicate-object expressions. These expressions are known as triples in RDF terminology. While the subject denotes the resource, the predicate denotes properties of the resource and expresses a relationship between the subject and the object (W3C 2014b). The simple concept of RDF makes is possible that "anyone can say anything about any topic" so that is can be used as a universal form for describing the world (Allemang and Hendler 2011, p. 323). The current RDF 1.1 specification was published in 2014 (W3C 2014c).

An important principle of RDF is resource identification (W3C 2014b) because different users may use the same names for different real world entities or different names for the same real world object. Only a clear identification of resources makes it possible to automatically integrate data from different sources (Yu 2011, p. 27). RDF uses Uniform Resource Identifiers (URIs) to identify resources instead of Uniform Resource Locators (URLs). The most important difference between URIs and URLs is that URIs do not have to represent a resource that is actually accessible via the network (dereferenceable) which

means that for example a URI beginning with “http://” does not have to represent a resource that is accessible via HTTP (Yu 2011, p. 28).

According to the RDF specification, the following holds for subjects, objects and predicates:

- A **subject** can be a URI named resource or a blank node
- An **object**, also called property value, can be a URI named resource, literal (string) or blank node.
- A **predicate**, also called property, must be a URI named resource.

URIs used by RDF consists of two components: An address and an identifier. When considering the URI *http://www.test.com#resource1*, the part preceding the #-character constitutes the address whereas the part succeeding the #-character is the identifier. RDF allows for shortening URIs used in triples by assigning namespaces to certain URIs.

In the following example (Table 3-2), RDF statements describing a camera and using literals as well as resources are depicted. The following namespaces were used:

- nikon: http://www.nicon.com/camera#
- camera: http://www.camera.com/#
- dbpedia: http://www.dbpedia.org/#
- rdf: http://www.rdf.com/#

Subject	**Predicate**	**Object**
nikon: Nikon_D300	rdf: is_a	camera:DSLR
nikon: Nikon_D300	camera: cameraModel	“D300”
nikon: Nikon_D300	camera: manufacturedBy	dbpedia:Nikon
nikon: Nikon_D300	camera: weight	“0.6 KG”

Source: Based on Yu (2011), p. 24

Table 3-2 Exemplary RDF Statements

RDF itself is an abstract model and does not refer to a file format or a particular language for encoding data but rather to the data model of representing information in triples (Allemang and Hendler 2011, p.59). Hence, RDF has several serialization formats. In practice, the formats Turtle, N-Triples, N-Quads, Json-LD, N3 and RDF/XML are used (Yu 2011). The latter was first standardized by the W3C in 1999 (W3C 1999). A collection of RDF statements spans a graph (W3C 2014b). As such, RDF-based data models are naturally more suited for certain kinds of knowledge representations than

relational models. Documents created with RDF are also called instance documents in Semantic Web terminology (Yu 2011, p. 151).

RDF defines at set of core terms/concepts to be used when creating RDF documents. The most frequently used concept "is a" assigns a resource to a specific class of resources. The namespace to be used when referring to RDF core terms is: http://www.w3.org/1999/02/22-rdf-syntax-ns#.

Ontology Languages

An ontology can be defined as "the manifestation of a shared understanding of a domain that is agreed between a number of agents and such agreement facilitates accurate and effective communication of meaning, which in turn leads to other benefits such as inter-operability, reuse and sharing" (Agarwal 2005). A shared understanding is reached by formalizing vocabularies of terms (ontologies) and specifying their meaning by describing relationships to other terms in the ontology (W3C 2012).

Although different ontology languages provide different degrees of expressiveness, they share the following minimal set of components (Corcho et al. 2006):

- **Classes**: Represent concepts used for grouping similar entities (correspond to entities in the Entity Relationship Model). Examples are city, hotel, car, bike, person, tree, and role. Classes in an ontology are usually organized in taxonomies through which inheritance mechanisms can be applied. For instance, it can be defined that *Table* is a sub class of *Furniture*.

- **Relations**: Represent a type of association between concepts/classes. Ontologies usually contain binary relations where the first argument is known as the domain of the relation and the second argument as the range. For instance, a binary relation *hasArrivalPlace* could have the concept *Travel* as its domain and the concept *Location* as its range. Relations are also called properties, whereby object properties and data properties can be distinguished. Object properties represent a relation between two classes, whereas data properties represent a relation between a class and a literal such as string, integer, data, etc.

- **Axioms**: Axioms are defined as sentences that are always true (Gruber 1993). They are normally used to represent knowledge that cannot be formally defined by the other components. In addition, formal axioms are used to verify the consistency of the ontology or to infer new knowledge (statements). An axiom could for example state that instances of class Person having the same *firstName*, *lastName* and *eMail* can be considered the same person.

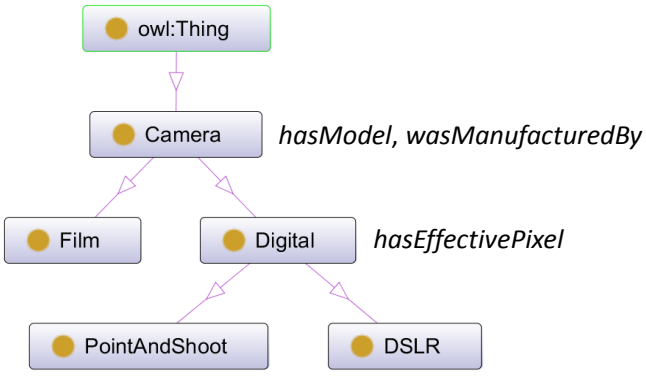
- **Instances**: Represent concrete elements or individuals in an ontology. For example, an instance of class *Person* could be *Person1* with *James* as *firstName*, *Rider* as *lastName* and *james.rider@domain.com* as *eMail*.

The vocabularies defined by ontologies are used in RDF-statements to enable an easy integration of statements originating from different, independent (web) sources (Yu 2011, p. 111). Therefore, RDF is used to create and describe instances with terms defined using ontology languages.

The Web Ontology Language (OWL) is currently the most popular and also the most powerful language for creating ontologies. The current version is OWL 2 which was standardized by the W3C in 2009. Another ontology language that can, with respect to their expressiveness, be seen as a subset of OWL, is RDF Schema (RDFS). Both ontology languages define a set of core terms that should be reused during ontology creation. This means that although RDFS is less powerful and in most cases not used for the definition of a new ontology, its terms can be reused when defining an ontology with OWL (Yu 2011, p. 155). The core vocabularies (ontologies) defined by RDFS respectively OWL are available under the following URIs:

- http://www.w3.org/2000/01/rdf-schema#
- http://www.w3.org/2002/07/owl#

Like RDF, all Ontology languages rely on the principle of resource identification. This means that all defined classes and relationships have to be uniquely identified via URIs. Only by adhering to this principle, unambiguity and hence an efficient data integration is possible. Continuing the example of statements about cameras, an ontology describing a common language for cameras could look like depicted in Figure 3-5:



Source: Based on Yu (2011), p. 111

Figure 3-5 A simple Ontology for the Camera Domain

The ontology describes the following facts: There exist resources called *Camera* with two sub-classes *Digital* and *Film*. The resource *Digital* in turn has two sub-classes *DSLR* and *PointAndShoot*. Besides the sub-class relationship, the ontology describes which properties can be used to describe certain classes of resources. In this case, resources of type *Camera* can be described with the properties *hasModel* and *wasManufacturedBy*, whereas resources of type *Digital* can be described with the property *hasEffectivePixel*. Since a sub-class relationship between *Digital* and *Camera* exists, all objects of type *Digital* can also be described with *hasModel* and *wasManufacturedBy*. For brevity, the domains and ranges of the properties are excluded at this point.

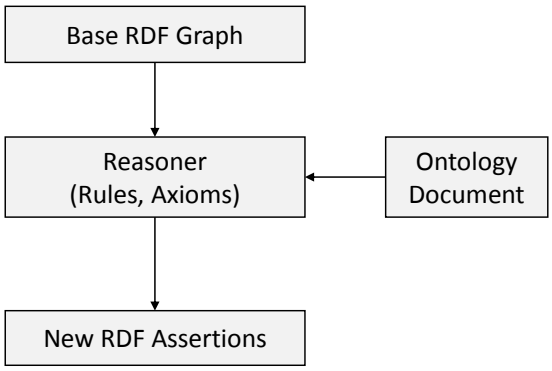
Since RDF is a universal format for describing things, also ontologies created with RDFS or OWL can also be transformed to an RDF document.

SPARQL

To retrieve data from (remote) RDF based data sources, an SQL-like query language called SPARQL was standardized by the W3C. Even though the syntax of SPARQL is similar to SQL, its underlying logic differs fundamentally. Every SPARQL query defines a graph pattern that is matched against the queried RDF-graph. Matched patterns are returned as the result. SPARQL queries can merge the results from an arbitrary number of RDF documents or SPARQL endpoints in one query. Besides the graph based logic, this is one of the major differences and advantages to SQL (Allemang and Hendler 2011, p. 61 ff.).

Reasoning

Reasoning refers to the process of deriving facts that are not explicitly expressed by a given instance document. As depicted in Figure 3-6, a reasoner is used to derive additional RDF statements from a given RDF graph by using the concepts defined in an ontology (Yu 2011, p. 471).



Source: Based on Yu (2011), p. 471

Figure 3-6 Principles of Reasoning

Typical reasoning tasks are the inference of sub-class relationships. For example, given an appropriate ontology, a reasoner can infer that a resource of type *Apple* also belongs to the resources of type *Fruit*. Using axioms defined in an ontology, a reasoner can also infer whether two collected resources actually describe the same thing. If, for instance, a data property *p* is defined as *functional*, which means that an instance of *p* can only belong to exactly one resource, a reasoner can infer, that, if another resource with the same data property is found, these two resources are the same. Since a complete coverage of all reasoning capabilities is out of the scope of this thesis, used axioms will be explained during the ontology development.

3.4.3 Concepts

Linked Data and the *Web of Data* are concepts that are closely related to the concept of the Semantic Web. The idea of Linked Data was originally proposed by Tim Berners-Lee and his Linked Data principles (W3C 2006) are considered to be the official and formal introduction to the concept (Yu 2011, p. 16). The Linked Data movement focuses on connecting datasets across the Web and can be seen as a subset of the Semantic Web concept, which is generally about adding meaning to Web content. Linking datasets is possible by creating statements (triples) about resources from two different sources. Further mechanisms, that can't be discussed here in detail, enable semantic web crawlers to follow each link to the source and collect additional statements about the referenced resource that were not available at the referencing source. Instead of having web pages pointing to each other via URLs (in the traditional web), one resource can point to another via URIs (Allemang and Hendler 2011, p. 6). The *Web of Data* is basically an interchangeable term for the Semantic Web: Once Linked Data is realized through Semantic Web technologies, the result is a Web of Data (Yu 2011, p. 17).

3.4.4 Architecture

SPARQL was developed not only to retrieve RDF data from one single source (like SQL) but to retrieve data from multiple different sources. The most important sources of RDF triples are RDF documents (serialized triples), XHTML webpages integrating RDF data through markup languages like RDFa, native triple stores that provide a SPARQL endpoint for querying it as well as relational databases that provide a SQL-SPARQL Bridge making it accessible to SPARQL queries. SPARQL has the ability to directly integrate different data sources in one query and merge the result (Allemang and Hendler 2011, pp. 55–59). A problem of solely using SPARQL to retrieve and analyze data is, however, that reasoning cannot be applied. Only those data that are directly available can be analyzed. This is why many semantic web applications first collect all triples from

relevant sources through SPARQL, save them into an internal triple store and apply reasoning to the data. Subsequent queries can now operate on the internal triple store instead. This provides the advantages of a richer data source and faster execution for subsequent queries. It can, however, also cause performance issues when large amounts of data have to be integrated (Allemang and Hendler 2011, p. 109). The prototype presented in this thesis uses an internal triple store. Figure 3-7 summarizes the explained semantic web architecture.

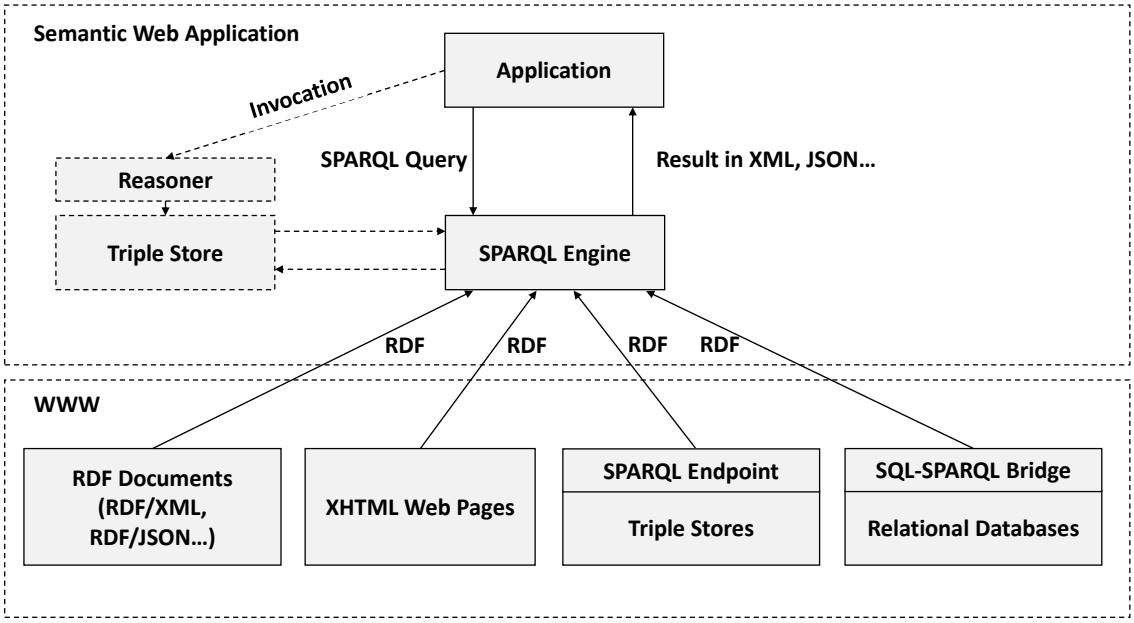


Figure 3-7 Semantic Web Architecture

3.5 Related Work

This thesis is not the first attempt to use semantic technologies for business process intelligence. Existing approaches, however, mainly focus on the analysis of control flow aspects of workflow models instead of event analysis. Only one approach could be found that explicitly deals with workflow event analysis based on ontologies (Haller et al. 2008). To the best of the knowledge of the author of this thesis, no approach exists in literature so far that deals with the systematic creation of a prototype for semantic process intelligence applying state of the art semantic technologies. In the following, identified related work will be summarized.

Based on the process specification ontology designed by GRÜNINGER (2004) and the MXML event exchange format, HALLER et al. (2008) present a process interchange ontology (oXPDL) based on the standardized XML Process Definition Language (XPDL). oXPDL explicitly models the semantics of XPDL and enables querying and reasoning over multiple models. In addition to a semantic representation of process

models, oXPDL also provides means to save process instance data. The proposed ontology is based on the Web Service Modelling Language (WSML), a special web ontology language focused on the description of semantic web services. In their publication, the authors provide several conceptual queries that are, however, not based on SPARQL, the standard for querying semantic web data. The provided download link to their ontology is not working anymore and a subsequent web search did not yield any concrete results for oXPDL. This thesis is partially based on the ideas presented in their research paper.

NICOLA et al. (2007) describe a method aiming at the reduction of ambiguity in process models produced by business experts, allowing for an easier translation into executable process models by IT experts. The method is based on the idea of semantic enrichment of process models. The enrichment is obtained by a semantic annotation of the process model elements based on a core business process ontology developed during their research (CBO).

FERREIRA et al. (2012) present an ontology-based approach for the automatic discovery of workflow activity patterns. The approach makes use of a mapping between the elements in a given process and the classes in a pattern ontology. A reasoner is used to automatically infer the existence of patterns while SPARQL queries are then used to retrieve the inferred patterns.

HERAVI et al. (2010) are leveraging ontologies for facilitating the creation of collaborative business processes. To achieve this, the authors present an ontology created with OWL for the ebXML Business Process Specification Schema (edBP), a business to business (B2B) process standard focusing on public processes and documents exchanged between trading partners in B2B transactions. The ontology can be used to capture and share the semantics of B2B processes, reducing the ambiguity of processes and exchanged information and hence facilitating the integration of processes between different business partners.

QUILITZ and LESER (2008) are showing how SPARQL Update queries can be used for complex event processing (CEP). They argue that an RDF-based event format provides a more flexible representation of heterogeneous events from distributed system compared to traditional formats due to its unambiguous semantics.

ZHU et al. (2010) designed an OWL ontology for representing events in the reservoir engineering domain. The ontology defines the major types of events and their properties. It also includes relations between the events such as cause relationships.

4 Ontology Development

The ontology development chapter starts with a description of the development process (section 4.1). Afterwards, the specification (section 4.2), conceptualization (section 4.3) and implementation (section 4.4) are presented.

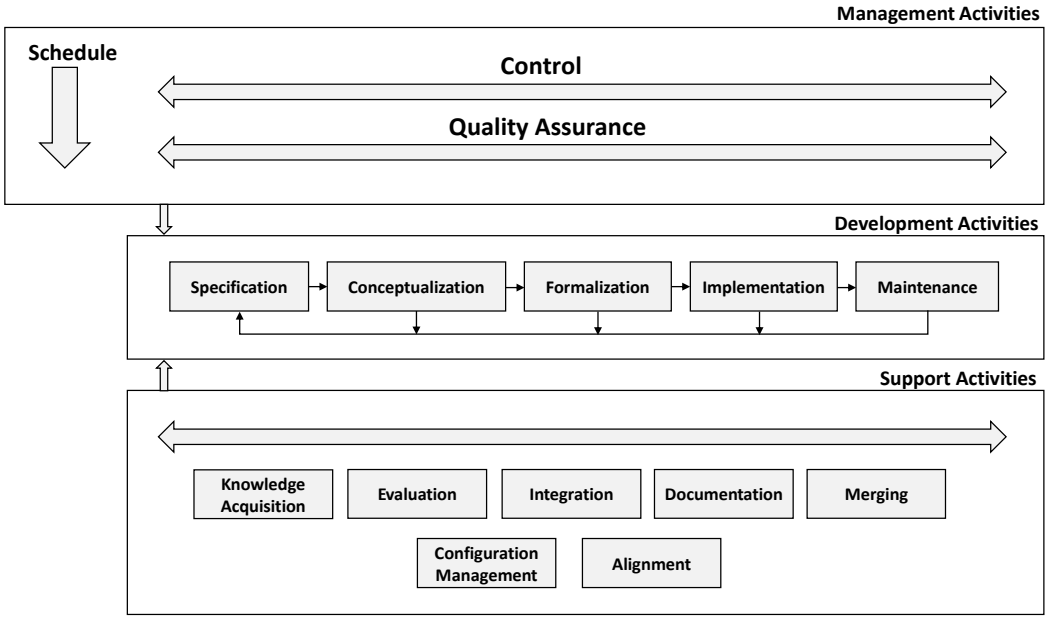
4.1 Development Process

Before creating a new ontology, three main questions should be considered (Corcho et al. 2003): (1) Which methodology should be used for building the ontology, (2) which ontology language should be used to implement it and (3) which tool should be chosen that best supports the building process. In the following, these questions are answered.

4.1.1 Methodology

In literature, different methodologies (like *On-To-Knowledge*, *CO4* or *Methontology*) for the development process exist. According to CORCHO et al. (2006), *Methontology* is the methodology that provides the most detailed descriptions of the steps to be performed while at the same focusing on a reasonable amount of steps. Hence, this methodology is chosen for the further development process.

Methontology subdivides into activities for Management, Development and Support as depicted in Figure 4-1. Management activities include scheduling, control and quality assurance. Due to the limited time scope of this thesis, management steps are not further considered.



Source: Based on Corcho et al. (2006), p. 12

Figure 4-1 Methontology Activity Types

Development-oriented activities are grouped into pre-development, development and post-development. During pre-development, an *environment study* identifies the problems to be solved with the ontology. An additional *feasibility study* should answer the question if it is possible and suitable to build the ontology. In the development phase, the *specification* activity states the purposes and scope of the ontology as well as the intended users. The *conceptualization* activity structures the domain knowledge in a meaningful way such as a collection of important terms and concepts that should be present in the ontology. During this activity, also the reuse of existing ontologies should be considered. The *formalization* activity uses the knowledge collected during the conceptualization phase and creates a formal model, while the *implementation* activity translates this model into a chosen ontology language by defining classes, properties and axioms. Post-development is about *maintenance* and *use* of the ontology. The support activities include a set of activities that should be performed during the development to support the development process. These are *knowledge acquisition* from domain experts, regular *evaluation* of results (e.g. through preliminary tests/queries), *integration* when already existing ontologies are reused, *documentation* and *merging* in case already existing ontologies are merged into a new and improved one. The *alignment* activity establishes links between involved ontologies preserving the original ontology without merging them. The *configuration management* activity records all versions of the documentation and of the ontology code to control changes (Corcho et al. 2006).

In sections 4.2, 4.3 and 4.4, the ontology development process will be described according to the steps *specification*, *conceptualization* and *implementation*. Since mature tools for the direct specification of an ontology in a chosen ontology language exist, the formalization step will be left out. The pre-development-phase was already covered in the introduction as well as in the foundation chapter. Necessary supporting activities will be mentioned when necessary.

4.1.2 Ontology Development Tool and Language

As ontology development tool, Protégé[1] was chosen. Protégé is currently the leading environment for ontology development. It is a free, open-source ontology editor written in Java. It was originally developed at Stanford University and is now supported by a community of developers and academic, government and corporate users (Yu 2011, p. 475). There exist two different variants of Protégé: WebProtégé is an ontology development environment that makes it possible to create, upload, modify and share ontologies for collaborative viewing and editing including a version history. Protégé Desktop is a feature rich ontology editing environment including different visualizations methods, reasoning and querying capabilities. Both versions support the latest version of OWL (Protege 2014) which is currently the most popular ontology language (Yu 2011, p. 155) and was chosen for the development process. At the time of this thesis, the desktop version (5.0) is better suited for the initial creation of an ontology while the web version is better suited to collaboratively discuss and improve existing work. WebProtégé, for instance, has no function to import existing ontologies into a new ontology. Hence, the desktop version was chosen for the development process.

The ontologies were published using GitHub[2]. WebProtégé could not be used since it doesn't provide any dereferencable URIs to the uploaded ontologies that can be used by the prototype.

4.2 Specification

The purpose of the ontology is to provide a general vocabulary that can be used to save and analyze business process event data, in specific audit trail event data (event data generated from well-defined executable workflow models). Hence, the ontology focuses on events generated from Business Process Management Systems (BPMS). Further, the ontology should enable event analysis of events originating from different systems with different state models. Hence, it should be general enough to capture events from every BPMS but detailed enough to provide a basis for useful business insights. To enable more meaningful queries, the ontology should incorporate background knowledge like information about the workflow model or contextual information about specific workflow instances (e.g. customer information or executing participant). The intended user group of the ontology are business analysts. It is assumed that this user group has a sound background knowledge about BPMS and its domain specific vocabulary.

[1] http://protege.stanford.edu/.
[2] https://github.com/sonaris/spi/.

A way to determine the scope of the ontology is to define a set of competency questions that instance data, based on the ontology, should be able to answer (Haller et al. 2008; Heravi et al. 2010; Uschold and Gruninger 1996). Since the major goal of the ontology is to provide a basis for event analysis, event-related questions will be in focus. Additional, event-independent questions and appropriate queries can be found in appendix C.

At least the following questions should be answerable through appropriate SPARQL queries. The questions are grouped according to their type into general, quality, time, control-flow and cross-system questions. A focus is set on time-related questions since these are of major interest in the context of BPI (see section 3.2).

General:

1. How many activity and workflow events from which system are available?

Quality:

2. How many of the opened instances of activity a / workflow w were completed?

Time:

3. What is the average, minimum and maximum turnaround, running and suspended time of activity a?
4. On which activities does participant p, in total, spend the most time on?
5. Which roles have, in total, the highest work load?

Control-Flow:

6. Which execution paths exist for workflow w and how often were they traversed?

Cross-System:

7. Which customers that participated in workflow w_1 from system s_1, also participated in workflow w_2 from system s_2?

4.3 Conceptualization

The conceptualization activity is about the collection and structuring of domain knowledge that will be used for the implementation of the ontology. To identify relevant existing knowledge, a literature review was conducted (see section 4.3.1). Sections 4.3.2 and 4.3.3 present the results of the conceptualization activity based on findings from the literature review and own work.

4.3.1 Information Sources

As part of the conceptualization, a literature review aiming at identifying requirements for a process intelligence event exchange format was conducted. Therefore, relevant literature sources, keywords and a reasonable time period had to be chosen (Webster and Watson 2002). To search for relevant literature in the area of business process management and workflow management, the following search query was used:

- (“event exchange format” OR “event format”) AND (process OR workflow)

The results were further constrained by excluding contributions that were published before the year 2006. Several search runs on common literature sources for the IS field revealed that Google Scholar (920) and Springer Link (92) return the most results (Google Scholar, Springer Link, EBSCO, ACM, IEE, AIS, Science Direct JSTOR were searched). Hence, these sources were chosen for further analysis. During a subsequent screening process of the search results (based on an analysis of title and abstract), relevant contributions were identified. In a final evaluation step including the removal of duplicates from the search results, a content analysis and an evaluation of references, 10 relevant sources could be identified. A short summary of the sources as well as a detailed description of the review process can be found in appendix B.

One contribution was identified to be particularly suited as a starting point for collecting requirements for an event ontology: *A Review of Event Formats as Enablers of Event-Driven BPM* by BECKER et al. 2012. The authors systematically compare the common (XML-based) event exchange formats like BPAF, MXML and XES and derive a list of general requirements for an event exchange format for edBPM. Despite a focus on edBPM instead of process intelligence, the requirements collected by the authors are equally valid for the scope of this thesis. This is due to the fact that edBPM also deals with the integration of business process events originating from different systems. The main difference is that edBPM focuses on automatic reactions to incoming events after the event analysis (Becker et al. 2012). Process intelligence can hence be seen as a subset of edBPM. Each requirement collected by the authors triggered a separate subsequent

literature review in the specific field, adding more specific literature to the already found as depicted in Figure 4-2. Section 4.3.2 summarizes the general requirements provided by the authors, whereas section 4.3.3 deals with the detailed requirements based on the initial and subsequent more specific literature reviews.

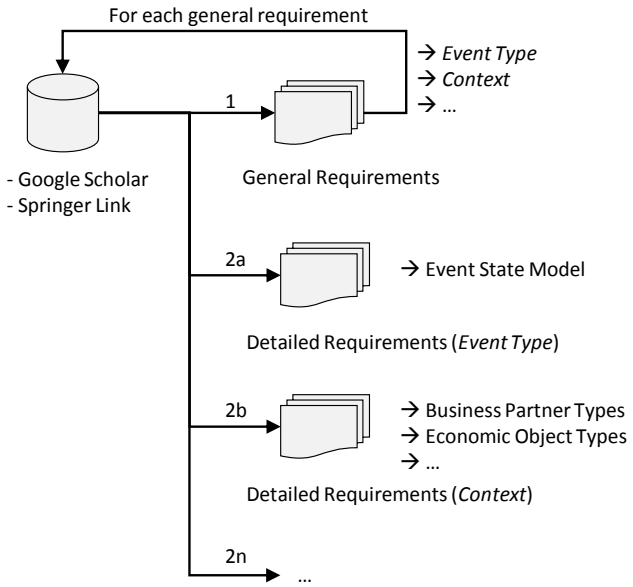


Figure 4-2 Iterative Literature Review Process

In addition to searching literature databases, public ontology repositories were searched for useful input. Based on recommendations by the W3C and a general web search using the search string *ontology repository*, the following relevant repositories could be identified: DERI Vocabularies[3], Schemapedia[4], SchemaWeb[5], WebProtege[6], ProtégéWiki[7], FIBO Semantics Repository[8] and a list of ontologies provided by the W3C[9].

4.3.2 General Requirements

BECKER et al. (2012) analyzed existing (XML-based) event exchange formats like BPAF, CBE or XES according to their weaknesses and strengths and extracted general requirements for an event exchange format that can be used in the context of event driven business process management (edBPM). The goal of edBPM is to provide a platform for exchanging and processing business events in order to turn insights into immediate actions in a (semi)automated manner. A crucial challenge in edBPM is the

[3] http://vocab.deri.ie/.
[4] http://schemapedia.com/.
[5] https://schema.org/.
[6] http://webprotege.stanford.edu/#List:coll=Home.
[7] http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library.
[8] http://www.edmcouncil.org/semanticsrepository/index.html.
[9] http://www.w3.org/wiki/Good_Ontologies.

communication of events among several event-producing and event-consuming systems (Becker et al. 2012). According to the authors, existing formats for event exchange do not sufficiently address the needs for edBPM, hence a list of requirements for an optimal event format is derived from existing formats. Due to the similar goals of edBPM and process intelligence (process intelligence can be seen as a subset of edBPM), the derived requirements are also suited for the purposes of the ontology.

The derived requirements address in particular the questions of what has happened (*identifier* and *type*), when did it happen (*time*), where did it happen (*origin*) under which circumstance did it happen (*context*), why did it happen (*cause*) and which effects may it impose (impact). In the following, each mentioned dimension will be described shortly and translated to general requirements for the ontology:

Identifier: BPM audit events typically contain data that is specific for a certain workflow or activity instance and allow for an unambiguous identification of single events. Thus, the ontology must allow for a unique and consistent identification (also for events from different systems).

Type: A classification of events allows for filtering or correlating events. Different event types are especially important to analyze the progress of a process or activity (is it suspended, active, completed etc.). A consistent analysis of events is only possible if standardized event types exist that can be mapped onto event types from different source systems. Hence, the ontology must contain a taxonomy of event types.

Time: Time is one of the most important dimensions since it allows for calculating time metrics such as turnaround time, active time or suspend time. Thus, the ontology must allow for exchanging the exact point in time when a certain event occurred.

Origin: The origin describes in which workflow or activity a certain event occurred. The more information about the origin of an event is available, the easier event tracing and interpretation becomes for a business analyst. Hence, the ontology should at least allow for assigning events to the workflow (instance) or activity (instance) they originated from. In an optimal case, the complete workflow definition should be part of the ontology.

Context: The event context has to consider all relevant business data, e.g. customers, products or orders. Since not all business situations can be ex-ante defined, predefining all contextual data is impossible. However, since an analysis of contextual data from different systems should be possible, at least those context data that can be standardized should be part of an ontology.

Cause: Audit events typically depend on a workflow model which determines causal relationships between the contained elements. The availability of causal relationships between events facilitates the extraction of time metrics by making a reconstruction of the event order based on timestamps unnecessary.

Impact: In the edBPM context, certain events shall cause certain actions in other systems and therefore have an impact. Since this component is not part of process intelligence as described in the foundation chapter, impact will not be further considered for the ontology.

4.3.3 Detailed Requirements

During the conceptualization phase, each dimension respectively general requirement triggered a separate literature review to identify more relevant sources in the respective area. The results were further analyzed regarding their potential to be represented in an ontology. The following sections summarize the detailed findings for the dimensions.

Identifier, Time and Cause

Every resource in RDF must have a unique identifier (URI). Every system that produces RDF statements will have its own name space like *http://www.system1.com/log.rdf* or *http://www.system2.com/log.rdf*. As long as each system makes sure that the given event Ids are unique within the systems scope, all event identifiers are unique. To facilitate the calculation of time metrics and avoid problems with different time zones, time will be measured in Unix time. Unix time is a system for describing instants in time, defined as the number of seconds that have elapsed since 00:00:00, Coordinated Universal Time (UTC), Thursday, January 1st 1970. Unix time is a single integer number which increments every second without requiring calculations of year, month, day, minute or second (Wikipedia 2014a). This in turn means that time metrics can be calculated on the basis of simple numbers instead of date or time strings which facilitates their calculation. The results can then be transformed back into a human readable form (months, days, hours, minutes or seconds). Since the number of seconds is, in most cases, not precise enough for storing events, the number of milliseconds will be used instead. All common programming languages like Java, C# or JavasScript provide built-in functions for returning the Unix time in milliseconds. Since integers on 32-bit systems do not provide enough digits for storing milliseconds since 1970, the long data type will be used instead. The dimension “cause” will be incorporated as a link between events that directly caused each other in the same workflow instance. This in turn facilitates again the calculation of metrics since the order of events doesn’t have to be reconstructed according to their

timestamps which makes SPARQL queries, extracting the metrics, shorter and easier to write.

Event Types

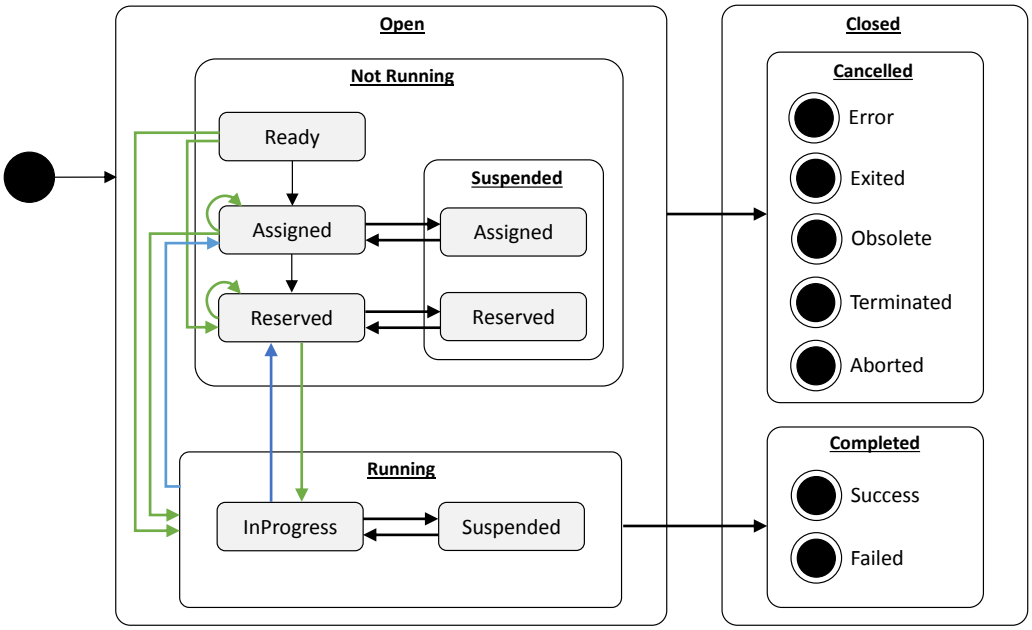
Business Process Management Systems support technical and human processes through the coordination of tasks, governing data exchanges and orchestrating application and service invocations. During the initialization and execution of process respectively workflow instances, multiple events occur (zur Muehlen and Swenson 2011). An activity can for example be "suspended" if no work is performed or "active" if an employee is working on a task.

The availability of process execution logs within BPMS can be traced back to the first commercial workflow systems of the late 1980s and early 1990s. Initially thought of as a way to enable troubleshooting and recovery of process instances in case of server failures, the use of process log files for analytics purposes was first highlighted by MCLELLAN (1996). The first attempt to standardize audit events came from the Workflow Management Coalition (WFMC) in 1996 when the Common Workflow Audit Data format (CWAD) was proposed. In 1999, a revision of CWAD was published enabling the integration of events stemming from different system what previously was not supported. CWAD, however, was only poorly accepted which is on the one hand due to its too fine granular event types including invocations of API functions of the system and on the other hand that it wasn't based on XML and had no prescribed serialization format. As a reaction, the mining XML format (MXML) was proposed by VAN DER AALST et al. (2005) and became the standard for the popular ProM process mining framework (zur Muehlen and Swenson 2011). According to ZUR MUEHLEN and SWENSON (2010), however, this format still has two shortcomings. First, limited possibilities to record arbitrary context data and second a too restrictive state model that makes a collection of events from different systems difficult.

According to the Workflow Reference Model, BPAF, as proposed by ZUR MUEHLEN and SWENSON (2011), is the current standard for exchanging process event data. BPAF is an XML-based interchange format for process audit events that is built upon a unified state machine for both processes and activities. It is able to accommodate multiple event originators and can map to the MXML format. A reason for the unification of the state models is the recognition that a process in one system may correspond to an activity in another system (zur Muehlen and Swenson 2011).

At the highest level, BPAF distinguishes between the two states *Open* and *Closed*. A process (or activity) is in the state *Open* if it can traverse through the state model through

internal or external impulses. A process (or activity) is *Closed* if it has reached a terminal state that it will not exit on its own. Each state is divided into a number of sub-states. The design consideration behind the sub-states is that different BPMS may expose a different level of granularity with regard to the events that can be observed. It is possible that one system records events that represent state changes in another system without access to the internal state system (e.g. a web service call). The BPMS might record the service as *Open* when the service is invoked and *Closed* when the service returns a result. It might also record whether the service invocation was successful from a business perspective (*Closed.Completed.Success*) or whether it did not deliver the desired results (*Closed.Completed.Failed*) (zur Muehlen and Swenson 2011). Figure 4-3 shows the state model created by ZUR MUEHLEN and SWENSON (2011).



Source: Based on zur Muehlen and Swenson (2011), p.5

Figure 4-3 BPAF State Model

The transitions shown are the most typical transitions in BPMS context but manual interventions and different system implementations may lead to additional transitions not depicted in the model (zur Muehlen and Swenson 2011). The fact that the transitions can differ from system to system also motivates the recording of current states instead of transitions which would also be conceivable (zur Muehlen and Swenson 2011).

The *Open* state is divided into the two sub-states *Running* and *NotRunning*. A process is in state *Running* if it is actively progressing towards its objective and is consuming resources, while a process in the state *NotRunning* can be scheduled for execution but does not progress towards its objective. The state *NotRunning* is further divided into the sub-states *Ready*, *Assigned* and *Reserved*, as well as *Suspended* (and its sub-states

Assigned and *Reserved*). These states accommodate the behavior of a BPMS with human activities that are handled through a work list. An activity that is ready for execution may be placed on the work lists of suitable performers (*Open.NotRunning.Assigned*) and one of these performers chooses to work on the activity instance (*Open.NotRunning. Reserved*). During this time, the activity instance may be barred from execution and is moved to the *Open.NotRunning.Suspended* sub-state. Transitions from *Assigned* to *Assigned* or *Reserved* to *Reserved* accommodate the reassignment of an activity or the delegation of an activity to another performer (zur Muehlen and Swenson 2011).

The *Closed* state is divided into the sub-states *Cancelled* and *Completed*. The *Completed* state represents the natural end of processing for a process or activity instance. It is further divided into (from a business perspective) successful and unsuccessful completions (*Success* or *Failed*). For example, a sales process may not lead to the favored signing of a contract. The distinction between *Success* and *Failed* allows for the quick aggregation of activity and process instances (zur Muehlen and Swenson 2011).

BPAF assumes that at the most basic level, a system will record *Open* and *Closed* states. But a system can choose to implement any number of sub-states and may choose to extend the state model with sub-states of its own. If an analytics system is presented with a BPAF event that is based on an extended state model, it can reduce the extended state until it arrives at a state it recognizes. For example, the extended state *Closed.Completed.Failure* would be reduced to *Closed.Completed* by a system that does not understand the extended state (zur Muehlen and Swenson 2011).

For the ontology creation, a slightly simplified version was chosen as depicted in Figure 4-4.

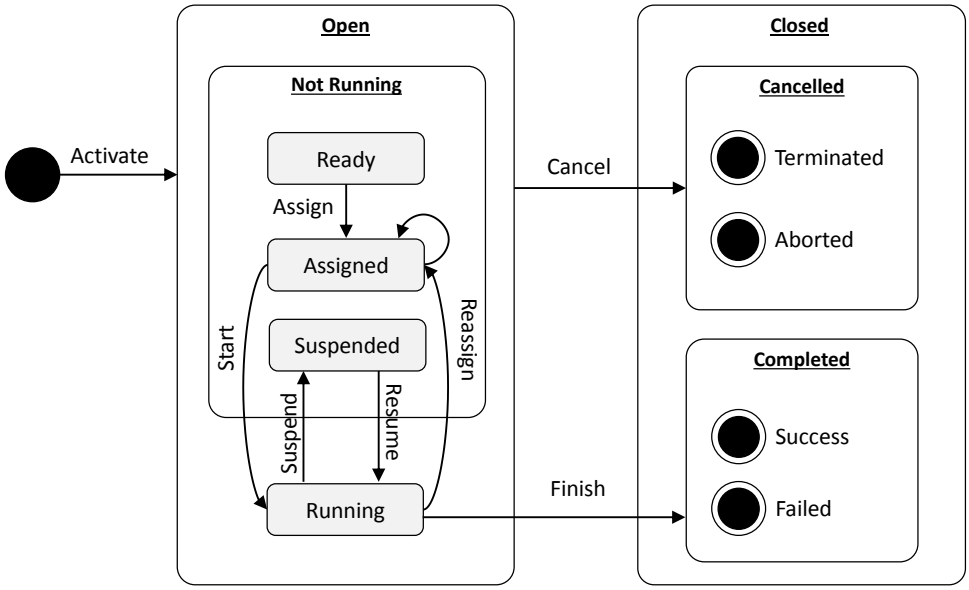


Figure 4-4 Chosen State Model

First of all, the *Cancelled* states were reduced to only *Terminated* or *Aborted*. Second, the distinction between a *Suspended* state for *NotRunning* and *Running* was removed and replaced by a single *Suspended* state within *NotRunning*. This was considered more reasonable since *Suspended* indicates that currently no work is performed, hence a *Suspended* state should only be available within *NotRunning*.

Origin

Incorporating a general workflow ontology allows for asking more complex question about the event data. For instance: How does the actual execution conform to the planned execution in the workflow model? Further, a workflow ontology allows for exchanging and analyzing the workflow model itself apart from the event analysis.

There exist basically two standardized exchange formats for workflows respectively process models in the market that can be used as a basis. The first and older one is the XML Process Definition Language (XPDL) which was first standardized in 2002 by the Workflow Management Coalition (WfMC) to interchange workflow process definitions between different workflow products, e.g. between different modeling tools and management suites. XPDL was designed as a universal serialization format for business processes, including the graphical location of elements. Hence, XPDL only defines a basic set of elements that is supported by most modeling languages (also BPMN). Extension possibilities however allow software providers to add additional functionality (Rücker 2008). According to the WFMC's website, XPDL is used today by more than 80 different products to exchange process definitions. As outlined in the WfMC reference architecture, XPDL is the recommended exchange format between process definition tools and enactment services (WFMC 2014b). The second exchange format is BPMN 2.0 XML. BPMN (Business Process Model and Notation), first published in 2005 by the Object Management Group (OMG), was originally designed only as a process definition language without any specific serialization format. Due to its immense popularity in the industry and academia – BPMN is the defacto standard for process modelling (Chinosi and Trombetta 2012) – a new version (2.0) was published in 2011 allowing for the design of executable processes (workflows) and included a serialization format. Several BPMS have already adopted this standard, among them Camunda BPM (Camunda 2014). The current version of XPDL (2.2), published in 2012, also supports the serialization of BPMN 2.0 models (WFMC 2014b).

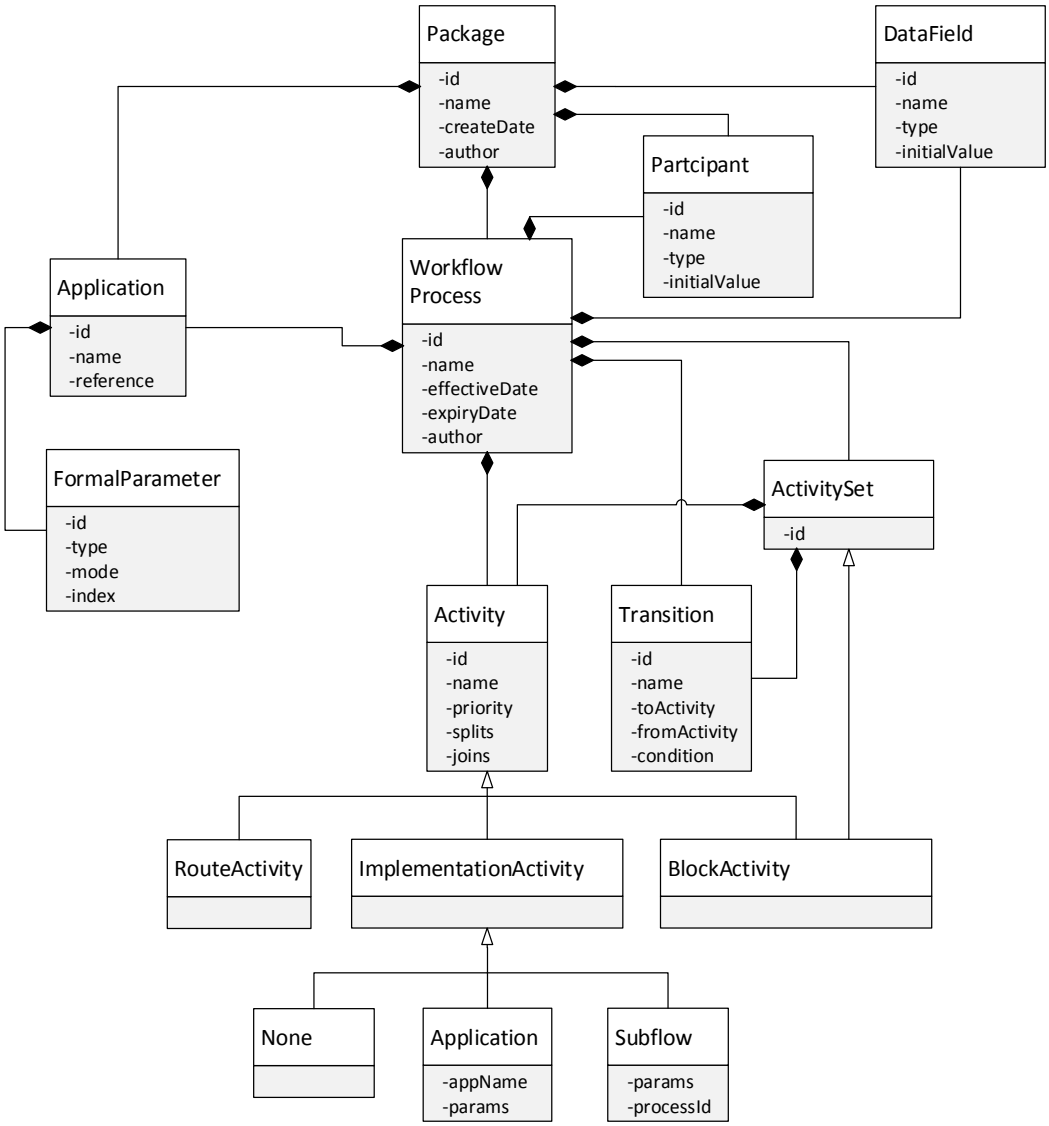
Due to its more universal approach and a broader acceptance in the industry, XPDL is chosen for the development of a workflow ontology that is compatible with most BPMS-systems. Since a complete conversion of the current XPDL 2.2 standard into an ontology was deemed unnecessary to show the general application principle, it was chosen that the

ontology should only reflect XPDL as presented in version 1. Additionally, the graphical aspect (location of elements) is neglected. It should be noted that the basic principles of XPDL haven't changed in version 2.2 since it is backward compatible with the first version, with minor exceptions (XPDL.ORG 2014).

In literature, one paper could be found that addressed the very same topic: The translation of XPDL into an ontological representation (Haller et al. 2008). According to the authors, the developed ontology would be available at the following address: m3pe.deri.ie/oXPDL/. The available download, however, did not contain the ontology. A subsequent web search for a downloadable version of oXPDL was not successful and the respective paper only provided limited information. Hence, the ontology had to be completely recreated. In the following, the basic elements of XPDL 1.0 will be described. A complete coverage of all elements can be found in the specification document[10] published by the WFMC.

The basic elements of XPDL and their relationships are depicted in Figure 4-5. The topmost entity is Package, which defines an entire application or a major system. A package consists of one or more workflow process definitions and supporting entities, which include participants (a list of human or system actors that perform key process roles), applications (programmatic components called by the process), and data fields (also known as workflow-relevant data). The package definition allows for the specification of a number of common process definition attributes, which are then applied to all individual process definitions contained and may be omitted from the individual process definitions. A workflow process definition consists of one or more activities and a set of transitions that define the control flow between the activities. Additional features include conditional transitions as well as exclusive (XOR) or parallel (AND) gateways, joins and splits which give XPDL reasonable expressive power. XPDL activities fall into three categories: route, block and implementation. A route activity performs no work but simply forwards control to the activity targeted by its outgoing transitions. A block activity is a set of activities having its own execution scope. The third activity type, implementation, has three implementation types: none, application and subflow. An activity with type none is manually controlled and its completion must be explicitly signaled to the workflow management system. An application activity calls an invoked application through an application agent. A subflow activity invokes another workflow process as a nested sub-workflow (Havey 2005; Norin 2002).

[10] http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf.



Source: Based on HAVEY (2005) and NORIN (2002), p. 14

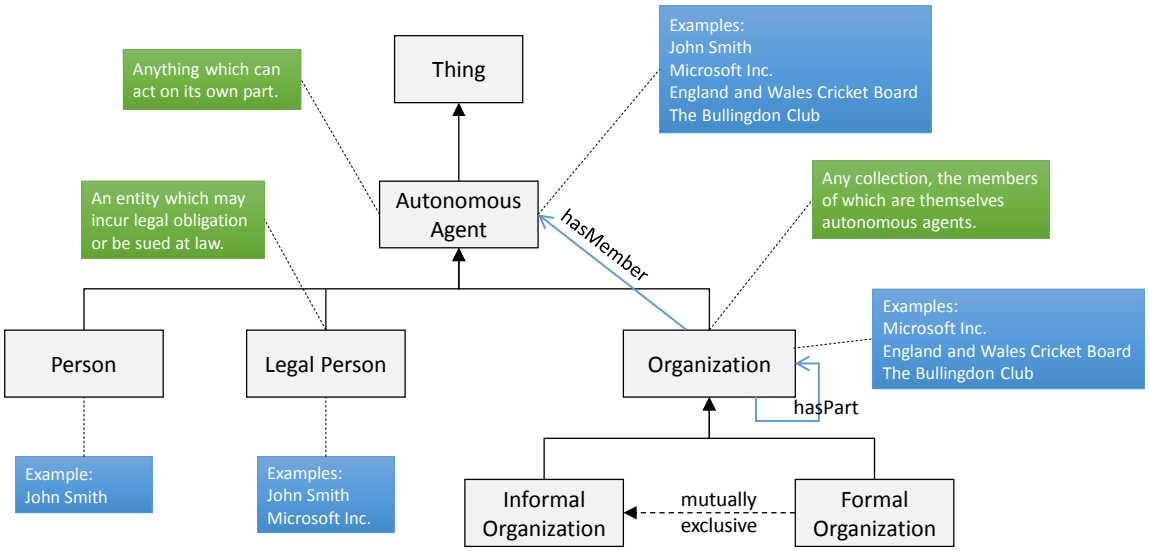
Figure 4-5 The XPDL Meta-Model

Context

Based on the results from the literature review and public ontology repositories, three different context types were identified that can be represented as an ontology in a reasonable manner:

Business Partner

For the business partner ontology, two useful information sources could be identified. The first source is the FIBO Semantics Repository. It already contains a Business Entity ontology for describing different actors in the financial industries (EDMCouncil 2014).



Source: Based on EDMCouncil (2014)

Figure 4-6 Business Entity Ontology

The second information source is SAP's definition of business partners as part of the SAP Supply Chain Management solution (SAP 2014). SAP describes business partners using two main concepts:

- Category: The three available categories are (1) natural person, (2) organization (e.g. company, department or club) and (3) group (e.g. married couple, shared living).

- Role: The available roles are (1) contact person, (2) employee, (3) organizational unit and (4) internet user.

It was decided to combine both views of a business partner by adding the business partner category *Group* as well as the concept of a role from SAP's view to the available ontology. The available roles from SAP were not considered as appropriate in the context of event analysis. It was decided that the ontology should at least contain the roles *Supplier*, *Employee* and *Customer* since these are the main roles interacting with BPMS from a company's point of view. Furthermore, *Legal Person* was removed from the ontology since it wasn't deemed necessary for analyzing events.

Economic Object

To be able to analyze for example orders or invoices that were handled during a workflow execution, it was deemed necessary to create an economic object ontology. Unfortunately, no suitable information about different types of economic objects (in literature or through available ontologies) could be found. The most similar but still not usable work (due to its too narrow focus) was an ontology for customer complaint management (Jarrar 2008).

To be able to include an analysis of economic objects, it was decided to create a simple ontology containing the following economic object types: *Contract*, *Invoice*, *Order* and *Complaint*.

System

Since (the same) workflows can be executed on different systems and by different users, it makes sense to create an ontology that provides means to describe them. Neither the literature review nor a review of existing ontologies could however yield any useful results. It was decided to create an ontology that, at least, contains the system type BPMS.

4.3.4 Considerations

In section 4.3.2, six different dimensions were identified that the ontology should cover. These are: Identifier, type, time, origin, cause and context. In section 0, the different event types to cover were detailed while section 0 described context elements that can be standardized as part of an ontology. The identified context types are *Business Partner*, *Economic Object* and *System*. To facilitate the implementation (due to better manageability) and improve the reusability of the results of this work, it made sense to create several independent ontologies that are connected by a central event ontology. In detail, the following ontologies are proposed:

- Workflow Ontology: Used for describing and exchanging workflow models.
- Business Partner Ontology: Used for describing and exchanging information about business partners such as customers or suppliers.
- System Ontology: Used for describing and exchanging information about systems (such as BPM System or CRM System) and users.
- Economic Object Ontology: Used for describing and exchanging information about economic objects such as orders, invoices or support cases.
- Event Ontology: Used for describing and exchanging audit trail events. The event ontology is the central ontology that has a connection to each of the other ontologies.

Another advantage of creating several ontologies that are connected by an event ontology is that it becomes easier to enrich the model, for instance by creating new context ontologies that just have to be connected via additional properties to the event ontology. The relationship between the five different ontologies is depicted in Figure 4-7.

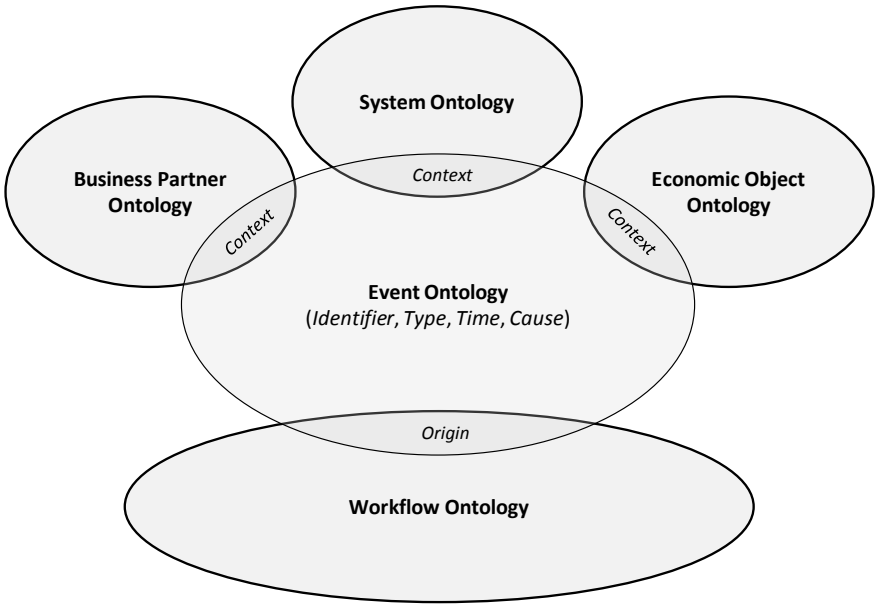


Figure 4-7 Relationship between the Different Ontologies

4.4 Implementation

During the implementation activity, the collected knowledge is transferred into ontologies. According to the considerations in section 4.3.4, a workflow, business partner, economic object and system ontology were created that are connected by an event ontology. In the following, each ontology including its classes, properties and (if necessary) axioms will be presented.

4.4.1 Workflow Ontology

As described in the conceptualization chapter, the workflow ontology is based on a transformation of the XPDL 1.0 specification into an ontological representation. In the beginning, the necessary classes had to be defined. As classes, the entities from the UML diagram shown section 4.3.3 were taken, resulting into the following representation in Protégé (see Figure 4-8). Subclass relationships are indicated by the folder structure. A special case is the *BlockActivity*. It is a subclass of both ActivitySet and Activity due to its multiple inheritance as shown in the UML diagram. The classes *FormalParameter* and *DataField* were not modeled since relevant context data should be described using the context ontologies.

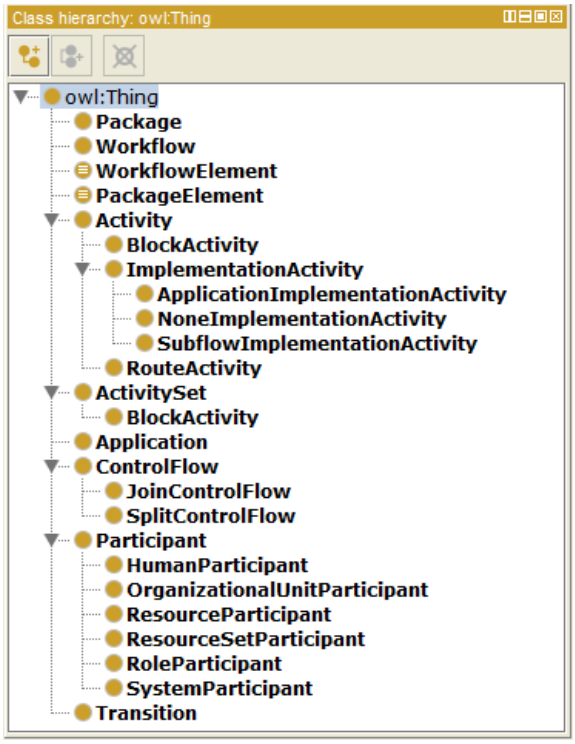


Figure 4-8 Workflow Ontology: Classes

The classes *WorkflowElement* and *PackageElement* were incorporated to illustrate simple reasoning capabilities. The first class contains the axiom "*equivalent (definedInWorkflow some Workflow)*" while the second class contains "*equivalent (definedInPackage some Package)*". If a reasoner is applied to an instance model of this ontology, all elements that have a definedInPackage respectively a definedInWorkflow relationship, will automatically be assigned to the PackageElement respectively WorkflowElement class even if this assignment has not been explicitly declared in the instance data.

In a second step, the object as well as data properties were defined. For the data properties, only a limited set of attributes from the specification was chosen. The motivation was to reduce complexity while at the same time enabling an understanding and analysis of the workflow. Table 4-1 and Table 4-2 list all properties including their respective domains and ranges.

Object Property	**Domain**	**Range**
definedInActivitySet	-	ActivitySet
definedInWorkflow	-	Workflow
definedInPackage	-	Package
hasIngoingTransition	JoinControlFlow	Transition
hasOutgoingTransition	SplitControlFlow	Transition
hasJoin	Activity	JoinControlFlow
hasSplit	Activity	SplitControlFlow
hasSourceActivity	Transition	Activity

(continued)

hasTargetActivity	Transition	Activity
performedByApplication	ApplicationImplementationActivity	Application
performedByParticipant	NoneImplementationActivity	Paticipant
precededByActivity	Activity	Activity
succeededByActivity	Activity	Activity

Table 4-1 Workflow Ontology: Object Properties

Data Property	Domain	Range
hasActivityName	Activity	String
hasApplicationName	Application	String
hasControlFlowType	ControlFlow	{“AND”, “XOR”}
hasPackageName	Package	String
hasParticipantName	Participant	String
hasTransitionCondition	Transition	String
hasTransitionName	Transition	String
hasWorkflowName	Workflow	String
hasWorkflowVersion	Workflow	String

Table 4-2 Workflow Ontology: Data Properties

The developed ontology was published with GitHub and is available at the following address: https://raw.githubusercontent.com/sonaris/spa/master/Ontologies/workflow.rdf.

4.4.2 Business Partner Ontology

As described in the conceptualization chapter, the business partner ontology is based on the ontology provided by the FIBO Semantics Repository as well as the representation of business partners in the SAP’s Supply Chain Management Solution. In the beginning the classes were again derived. The ontology is designed such that a new business partner instance always belongs to a subclass of *AutonomousAgent* as well as to a subclass of *Role*.

Figure 4-9 Business Partner Ontology: Classes

In a subsequent step, the object and data properties were derived.

Object Property	Domain	Range
hasMember	Organization	Person
hasOrganizationPart	Organization	Organization

Table 4-3 Business Partner Ontology: Object Properties

Data Property	Domain	Range
hasEmailAddress	AutonomousAgent	String
hasFirstName	Person	String
hasLastName	Person	String
hasOrganizationName	Organization	String
hasGroupName	Group	String
hasAddressCountry	AutonomousAgent	String
hasAddressCity	AutonomousAgent	String
hasAddressStreet	AutonomousAgent	String
hasAddressHouseNumber	AutonomousAgent	String

Table 4-4 Business Partner Ontology: Data Properties

Since event data from different independent event sources shall be integrated, it made sense to add axioms that can be used to decide whether certain instances from different sources describe the same real world entity. To enable inferencing over persons, the following axiom was added (Figure 4-10).

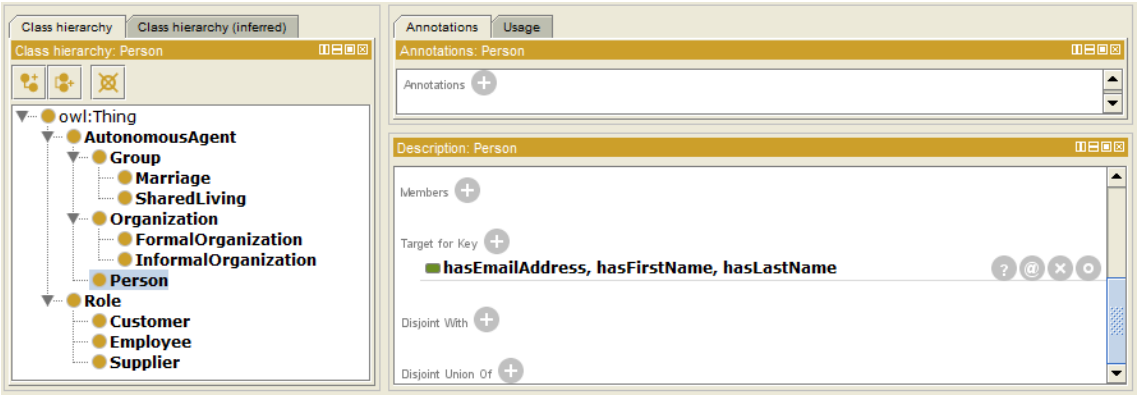


Figure 4-10 Creation of a Target for Key-Axiom in Protégé for class Person

The axiom defines the following fact: Whenever two instances of class *Person* are found that have the same email address, first name and last name, then these instances are considered the same if a reasoner is applied. The detection of such an inference leads to a new *"owl:same as"*-relationship between the respective instances. This type of relationship belongs to the core vocabulary of OWL and is added automatically. More complex inference mechanisms are possible and also recommendable for real life cases. However, this example is suitable to demonstrate inferencing over instances from different sources.

The developed ontology was published with GitHub and is available at the following address: https://raw.githubusercontent.com/sonaris/spa/master/Ontologies/businesspartner.rdf.

4.4.3 Economic Object Ontology

The economic object ontology is used to describe important objects that were handled during the execution of a workflow. For this thesis, only the classes *Complaint*, *Contract*, *Invoice* and *Order* were included (see Figure 4-11).

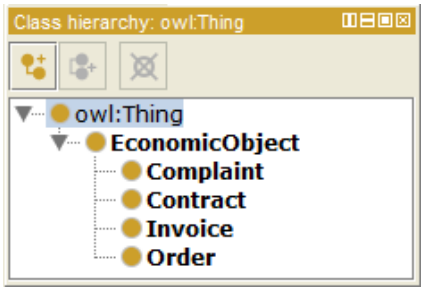


Figure 4-11 Economic Object Ontology: Classes

Each sub class was assigned a simple data property to identify the object.

Data Property	Domain	Range
hasInvoiceNumber	Invoice	Integer
hasOrderNumber	Order	Integer
hasContractNumber	Contract	Integer
hasComplaintNumber	Complaint	Integer

Table 4-5 Economic Object Ontology: Data Properties

To allow reasoning over economic objects, a Target-for-Key axiom was added to each object type like depicted in Figure 4-12.

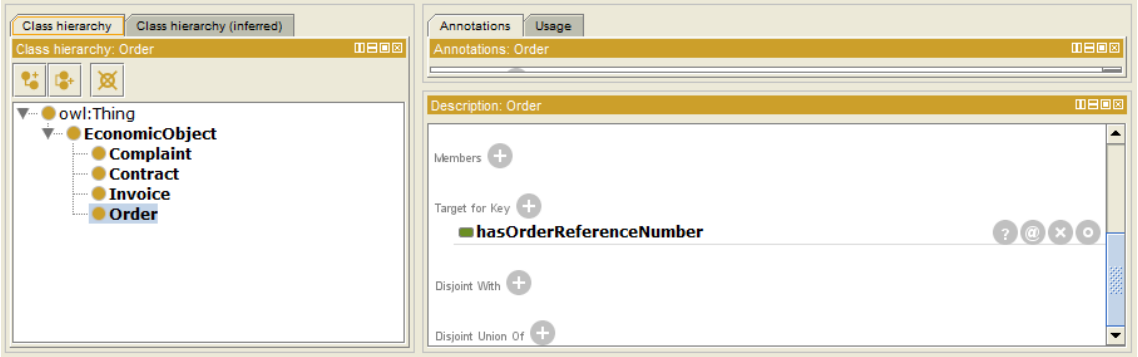


Figure 4-12 Creation of a Target for Key axiom in Protégé for the class Complaint

The developed ontology was published with GitHub and is available at the following address: https://raw.githubusercontent.com/sonaris/spa/master/Ontologies/economicobject.rdf.

4.4.4 System Ontology

The system ontology serves the purpose to save information regarding systems and user accounts that generated events. A comprehensive taxonomy of different system types could (theoretically) be included. So far, only the types *BPMSystem* and *CRMSystem* are prescribed since they are sufficient for illustrating the general principle.

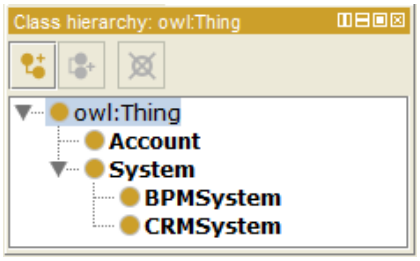


Figure 4-13 System Ontology: Classes

The object and data properties defined in the system ontology are described in Table 4-6 and Table 4-7.

Object Property	Domain	Range
isPartofSystem	Account	System

Table 4-6 System Ontology: Object Properties

Data Property	Domain	Range
hasSystemName	System	String
hasSystemDescription	System	String
hasAccountUserName	Account	String
hasAccountPassword	Account	String

Table 4-7 System Ontology: Data Properties

It would be possible to further extend the system ontology to be able to describe real world persons that are connected to the accounts. For purpose of this thesis, this was however not deemed necessary.

The developed ontology was published with GitHub and is available at the following address: https://raw.githubusercontent.com/sonaris/spa/master/Ontologies/system.rdf.

4.4.5 Event Ontology

Basic Elements

At the beginning, a basic event ontology was created that was independent from the other ontologies. This ontology only focuses on the types of events that can be created. Additionally to the event types discussed in the conceptualization chapter, the ontology

covers created workflow as well as activity instances. The hierarchical relationships of the different event types are expressed through different sub-class relationships as depicted in Figure 4-14. This representation facilitates later SPARQL queries (see chapter 5) since different sub types can be queried by just using one higher level event type.

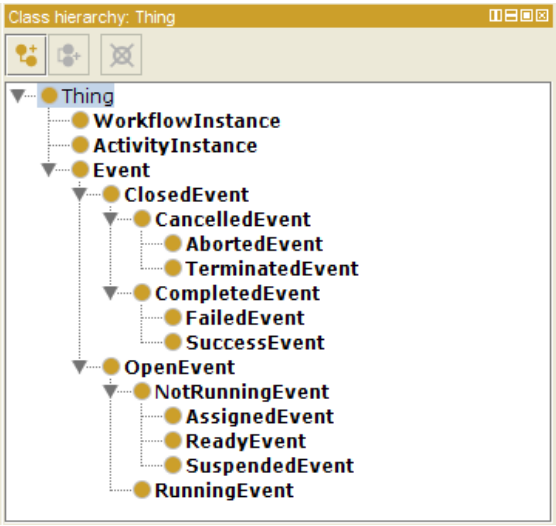


Figure 4-14 Event Ontology: Basic Classes

Additionally, different object properties were created to connect events.

Object Property	Domain	Range
partOfWorkflowInstance	ActivityInstance	WorkflowInstance
originatedFromWorkflowInstance	Event	WorkflowInstance
originatedFromActivityInstance	Event	ActivityInstance
precededBy (transitive)	Event	Event
succeededBy (transitive)	Event	Event
causedBy	Event	Event
caused	Event	Event
generatedBySystem	WorkflowInstance	System

Table 4-8 Event Ontology: Object Properties

Figure 4-15 depicts the logic behind the different elements and object properties. Since a unified state model for workflow and activities is used, events can be generated from a workflow instance or an activity instance. Whether both types of events are actually generated depends on the specific BPMS. If an event belongs to a specific workflow instance, it has to be connected to this instance (via *originatedFromWorkflowInstance).* In case it originated from an activity instance, it should be connected to this activity instance (via *originatedFromActivityInstance*). Workflow events are not mandatory. If only activity events are available, workflow events can be reconstructed from them. For example a suspended event in an activity instance always corresponds to a suspended

event in the workflow instance. Providing events for workflow instances just facilitates the analysis of workflows.

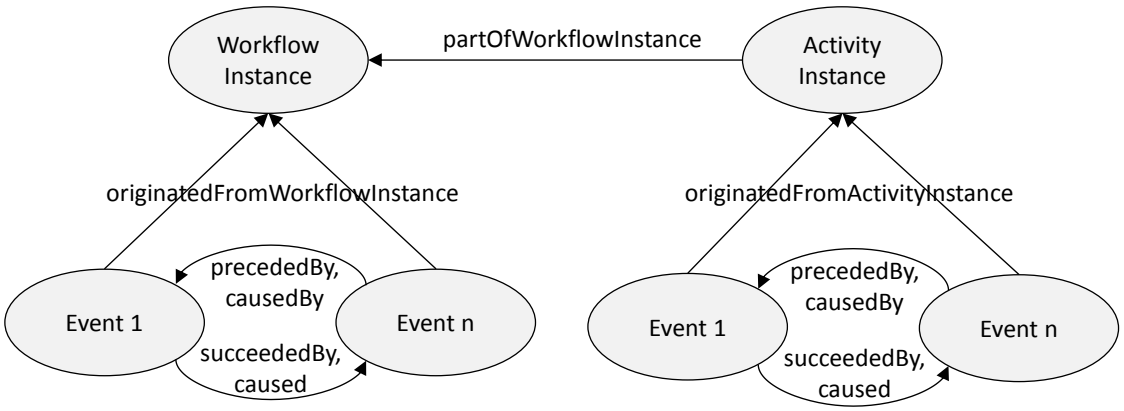


Figure 4-15 Event Ontology: Relation between Elements

CausedBy and *caused* connect the different events according to their cause-relationship, where one event can only have one cause-relationship. *PrecededBy* respectively *succeededBy* are used for a similar purpose but connect events according to their occurrence in an activity or workflow instance. This means that one event can have an arbitrary number of preceding or succeeding events. *CausedBy* and *caused* respectively *precededBy* and *succeededBy* are defined as inverse properties which enables a reasoner to infer the other direction of the connection if only one direction is available. Additionally, *precededBy* and *succeededBy* are defined as transitive. Transitivity has the following meaning: If a property p is defined as transitive and there exists a resource relationship $A - p \rightarrow B - p \rightarrow C$ (A is connected to B and B to C using p), a reasoner can infer that there also exists a relationship $A - p \rightarrow C$. This makes it easier to answer the following question: What are all events preceding/succeeding event x.

The basic event ontology just contains one data property: *hasTimeMilli*. As described in the conceptualization chapter, this property is used to save a timestamp in the Unix time format.

Data Property	Domain	Range
hasTimeMilli	Event	Long

Table 4-9 Event Ontology: Basic Data Properties

Integration of other Ontologies

To integrate existing ontologies, import statements for each ontology that should be referenced, have to be created in the event ontology. Protégé assists the user in creating such statements. A semantic application will later recognize the imports and

automatically download the ontologies from the respective web locations. In this case, the respective URLs from the GitHub repository were taken. In the following, the additional object properties are introduced that connect the event ontology to the other ontologies.

Adding a connection to the system ontology:

The system ontology is integrated by adding a connection between workflow instances and the system on which they were executed as well as by adding a connection between a produced event and its producer (Account) (see Table 4-10). The second property also provides means to analyze how the planned execution corresponds to the actual execution when e.g. a different user as planned executed an activity.

Object Property	Domain	Range
generatedBySystem	WorkflowInstance	System
producedByAccount	Event	Account

Table 4-10 Event Ontology: Object Properties connecting the System Ontology

Adding a connection to the workflow ontology:

The workflow ontology is integrated by adding the three properties listed in Table 4-11. The first property connects workflow instances with the workflow definition that defined its execution, the second respectively for activities. The third property is used to connect an Account (which is part of a BPMS) to the participant defined in the workflow model (usually a role).

Object Property	Domain	Range
definedByWorkflow	WorkflowInstance	Workflow
definedByActivity	ActivityInstance	Activity
belongsToParticipant	Account	Participant

Table 4-11 Event Ontology: Object Properties connecting the Workflow Ontology

Integration of the Business Partner and Economic Object Ontology:

To integrate the remaining ontologies, two further object properties were introduced that connect the workflow instance with the economic object and business partner ontology. For adding context, the workflow instance was chosen since in most cases, the context of a workflow instance and not of an activity instance or event are relevant. Also common BPMS like Stardust (see section 6.2.3) save context data per workflow instance. Typical questions that can be answered with this connection are: Which business partner took part in a certain workflow or which customer that ordered a product through workflow 1 then complained about it through workflow 2?

Object Property	Domain	Range
hasBusienessPartnerContext	WorkflowInstance	AutonomousAgent
hasEconomicObjectContext	WorkflowInstance	EconomicObject

Table 4-12 Event Ontology: Object Properties connecting the remaining Ontologies.

Figure 4-16 depicts the combined event ontology including the most important object properties.

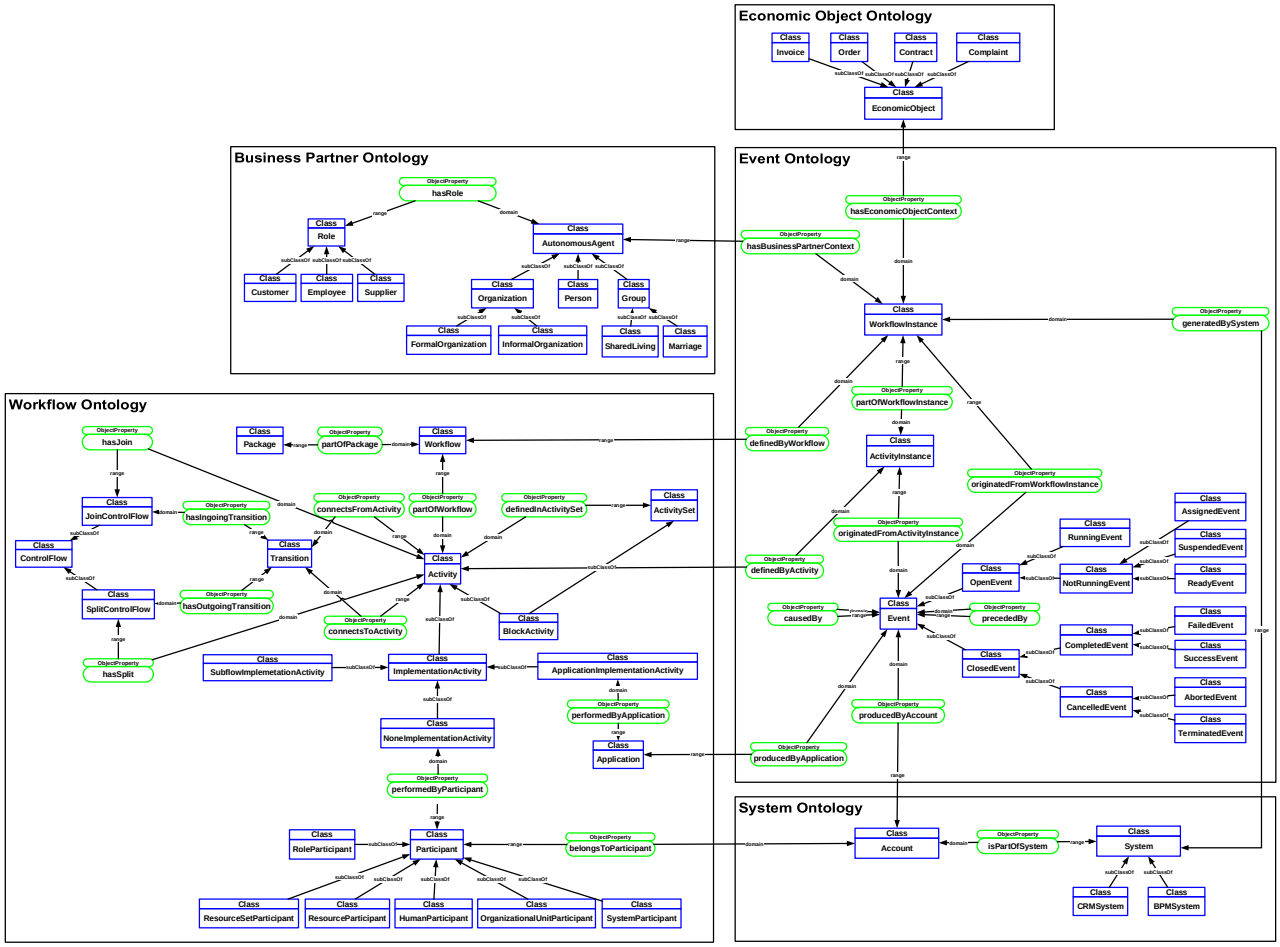


Figure 4-16 All Ontologies Combined

The combined event ontology contains 56 classes, 23 object properties and 27 data properties. The ontology was published with GitHub and is available at the following address: https://raw.githubusercontent.com/sonaris/spa/master/Ontologies/event.rdf.

5 Ontology Demonstration

In the following, the described ontology will be demonstrated by first generating synthetic event data (section 5.1 and 5.2) and then performing SPARQL queries on the data for answering the defined competency questions (section 5.3).

5.1 Synthetic Process Models and Event Data

To demonstrate the event ontology and show its potential in context of process intelligence, the following hypothetical example processes were developed. To create appropriate example event data based on the processes, a java-based event generator was used (see section 5.2). The process models are only used for illustration purposes and are not executed on a real BPMS.

Order Workflow

The *Order Workflow* is executed on the hypothetical system *Synthetic1*. It is a completely automatic workflow with no human intervention. It is triggered by a new order email that was, for instance, generated by a customer through a web form. Based on the provided information, the *NewOrderApp* creates a new order object (Create New Order) and passes it on the verification step (*Verify Order*) where the *OrderVerificationApp* analyses whether the order can be fulfilled as requested or not. An appropriate answer is sent accordingly in the subsequent step (*Send Positive Answer* or *Send Negative Answer*) using the *MailAppPositive* repectively *MailAppNegative*. In case the order can be fulfilled, it is forwarded by the *ForwardOrderApp* to another system dealing with the fulfilment process (*Forward Order*). The last activity files the order.

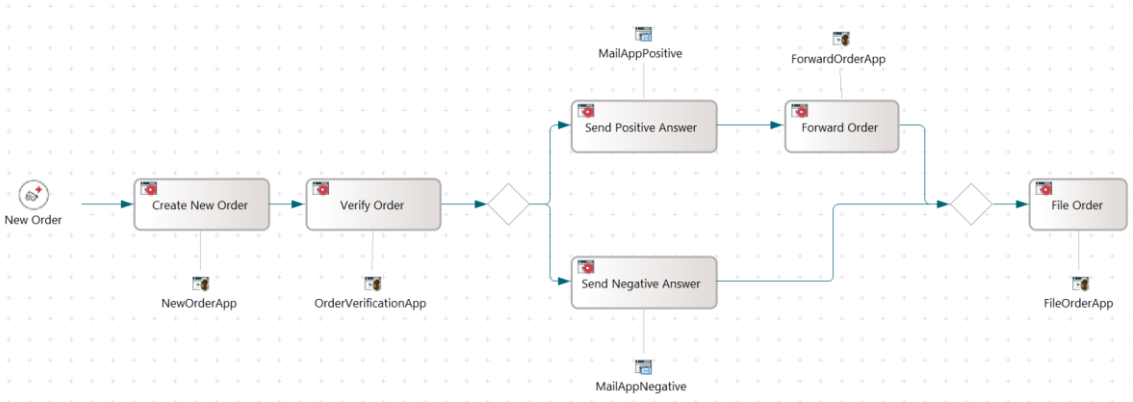


Figure 5-1 Order Workflow

Two example workflow instances were created. The first execution has the following order of performed activities: *Create New Order, Verify Order, Send Positive Answer, Forward Order and File Order*. The second execution order is: *Create New Order, Verify*

Order, *Send Negative Answer* and *File Order*. For each activity instance, the following events were created: *Open.Running* and *Closed.Completed*. Also for each workflow instance, the events *Open.Running* and *Closed.Completed* were created. Additionally, context was added to each workflow instance: Workflow Instance 1 handles an order by customer *Michael Champs* with email address *michael.champs@domain1.com* while workflow instance 2 handles a complaint by customer *Adam Williams* with email address *adam.williams@domain2.com*. Also, two order objects with *OrderReferenceNumber* 1 and 2 are attached to the respective workflow instances.

Complaint Handling Workflow

The *Complaint Handling Workflow* is executed on the hypothetical system *Synthetic2*. It is started manually by an employee of role *Service Agent* and initiated for instance by a customer phone call. An *Analyst* analyses the case and has to decide whether he can provide a solution (*Provide Solution*) or has to notify the customer that the problem cannot be solved (*Notify Customer*). When one of the two is done, the case will automatically be filed by the *FileCaseApp*.

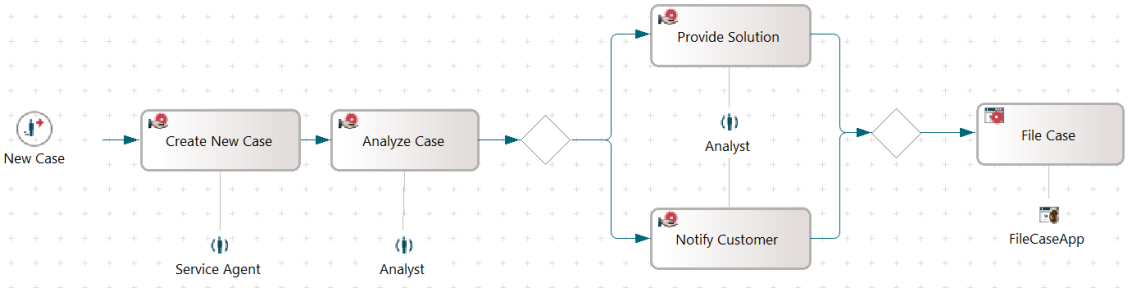


Figure 5-2 Complaint Handling Workflow

Two example workflow instances were created. The first execution has the following order of performed activities: *Create New Case*, *Analyze Case*, *Provide Solution, and File Case*. The second execution order is: *Create New Case*, *Analyze Case*, *Notify Customer* and *File Case*. For each manual task (activity instance), the following events were created in the given order: (1) *Open.NotRunning.Assigned*, (2) *Open.Running*, (3) *Open.NotRunningSuspended*, (4) *Open.Running* and (5) *Closed.Completed*. For each workflow instance, only the events *Open.Running* and *Closed.Completed* were created. Additionally, context was added to each workflow instance: Workflow Instance 1 handles a complaint by customer *Gregory Mitchel* with email address *gregory.mitchel@domain.com* while workflow instance 2 handles a complaint by customer *Adam Williams* with email address *adam.williams@domain.com*. The second customer is the same customer that took part in the order workflow. Also, two complaint objects with *ComplaintReferenceNumber* 1 and 2 are attached to the respective workflow

instances. Two user accounts *service_agent1@domain.com* with role *Service Agent* and *analyst1@domain.com* with role *Analyst* were created that are executing the workflow.

5.2 Synthetic Event Data Generation and Querying

In a first attempt, it was tried to generate and query synthetic event data by solely using the functionality provided through Protégé. After testing several small examples, however, it turned out that the creation of instance data in Protégé is too time consuming and error prone especially when the instances and their properties are changed many times which was the case at the beginning. Further, the SPARQL editor, provided by the current version of Protégé, has a major limitation: When reasoning is applied to the instance data created with Protégé, the additional inferred statements are not included in the query result. Additionally, the SPARQL-editor has no undo or redo functionality which makes systematic query development difficult. Both reasons prohibited the further use of Protégé for the ontology evaluation.

To be able to generate synthetic event data in an efficient manner, a java-program was developed that could, based on definitions in Excel files, automatically generate the necessary RDF data including inferred statements. The program was later integrated as the synthetic import into the final prototype. Since the prototype implementation is covered in a separate chapter, the specifics of the event data generation won't be explained in detail here (for more information see section 6.3.2). In total, three Excel-files for the *Order* and three Excel files for the *Complaint Handling* workflow were created from which the latter can be found in appendix E.

To be able to develop and execute SPARQL queries in an efficient manner, it was decided to set up new SPARQL endpoint with the created RDF-file as the data source. Different free SPARQL endpoints are available whereby *Jena Fuseki*[11] was chosen. Jena Fuseki is a part of the Apache Jena Semantic Web framework and provides an easy to use interface, query syntax checking and undo/redo functionality. All queries presented in the following were executed successfully via Fuseki. A small tutorial how to use Fuseki for executing queries can be found in appendix H.

5.3 Answering the Competency Questions

During the specification phase, a number of competency questions were defined which the developed ontology should be able to answer. By performing SPARQL queries on the generated example data, each question is answered in the following. The result of the query is shown in the respective table below. Due to space restrictions, not every

[11] Download: http://jena.apache.org/download/index.cgi.

SPARQL query can be explained in detail. For better readability, activity and workflow names were used as filter criteria instead of the full URIs.

The query for answering the question of how many activity and workflow events are available, consists of two sub-queries joined together by the using the "."- operator. The key word OPTIONAL makes sure that, even in case no activity or workflow events are available, a result will be returned for each system and these results can be joined using the system names. The SPARQL query engine automatically joins those rows together that have the same column names. In this case, *?s* and *?sName* are used for this process.

```
SELECT ?sName ?aEvents ?wEvents
WHERE{
  {
   SELECT ?s ?sName (COUNT(?event) as ?aEvents)
   WHERE{
     ?s a system:System.
     ?s system:hasSystemName ?sName.
     OPTIONAL{
       ?wi event:generatedBySystem ?s.
       ?ai event:partOfWorkflowInstance ?wi.
       ?event event:originatedFromActivityInstance ?ai.
     }
   }
   GROUP BY ?s ?sName
   ORDER BY ?s ?sName
  }
  .
 {
   SELECT ?s ?sName (COUNT(?event) as ?wEvents)
   WHERE{
     ?s a system:System.
     ?s system:hasSystemName ?sName.
     OPTIONAL{
       ?wi event:generatedBySystem ?s.
       ?event event:originatedFromWorkflowInstance ?wi.
     }
   }
   GROUP BY ?s ?sName
   ORDER BY ?s ?sName
 }
}
```

?sName	?aEvents	?wEvents
Synthetic1	18	4
Synthetic2	34	4

Query 1 Answering Competency Question 1

It can be seen that 18 activity and 4 workflow events were generated by the first system and 34 activity and 4 workflow events by the second system.

To find out how many instances of a certain workflow or activity were actually completed, one has to compare the events of type *OpenEvent* and *CompletedEvent*. For this, the completed event has to be encapsulated into an *OPTIONAL* statement since not every workflow instance includes this type of event. The following query illustrates the principle by analyzing the workflow instances of workflow *New Order*. It can be seen that both workflow instances were completed.

```
SELECT (COUNT(?eventOpened) as ?numberOpened)
       (COUNT(?eventCompleted) as ?numberCompleted)
WHERE
{
  ?w a workflow:Workflow; workflow:hasWorkflowName "New Order".
  ?wi event:definedByWorkflow ?w.
  ?wi event:generatedBySystem ?s.
  ?s system:hasSystemName ?sName.

  ?eventOpened event:originatedFromWorkflowInstance ?wi; a event:OpenEvent.

  OPTIONAL{
    ?eventCompleted event:originatedFromWorkflowInstance ?wi; a event:CompletedEvent.
  }
}
GROUP BY ?s ?sName ?w ?wName
```

?numberOpened	?numberClosed
2	2

Query 2 Answering Competency Question 2

In order to analyze activities instead of workflows, *originatedFromWorkflowInstance* has to be exchanged by *originatedFromActivityInstance* and *definedByWorkflow* and *hasWorkflowName* by its activity counterparts.

Another important class of queries is that for the calculation of time metrics (competency question 3). The developed ontology can be used to calculate all standard metrics like turnaround time, running/active time or suspend time. The following query calculates the turnaround time of activity *Analyze Case* from the *Complaint Handling* workflow. The idea behind the query is that first all open-events and closed-events are collected. From these, the earliest open and the latest closed events are extracted and compared.

```
SELECT (AVG(?tTime) as ?tTime_avg) (MIN(?tTime) as ?tTime_min)
       (MAX(?tTime) as ?tTime_max)
WHERE{
  SELECT ?a ?ai ((?end-?start) as ?tTime)
  WHERE{
    SELECT ?a ?ai (MIN(?tOpen) as ?start) (MAX(?tClosed) as ?end)
    WHERE {
      ?eOpen event:originatedFromActivityInstance ?ai; a event:OpenEvent;
        event:hasTimeMilli ?tOpen.
      ?eClosed event:originatedFromActivityInstance ?ai; a event:ClosedEvent;
        event:hasTimeMilli ?tClosed.
      ?ai event:definedByActivity ?a.
      ?a workflow:hasActivityName "Analyze Case".
      ?a workflow:partOfWorkflow ?w.
      ?w workflow:hasWorkflowName "Complaint Handling".
    }
    GROUP BY ?a ?ai
  }
}
```

?tTime_avg	?tTime_min	?tTime_max
43500.0	42000	45000

Query 3 Answering Competency Question 3 (Turnaround Time)

The result shows that the average turnaround time amounts to 43500 milliseconds whereas 42000 is the shortest and 45000 the longest turnaround time.

To calculate the running/active time, a different approach can be used. The following query searches for all pairs of events where a running event is succeeded by an arbitrary event. All time deltas are summed up for each activity instance. From theses sums, the average, minimum or maximum running time for an activity can be calculated.

```
SELECT (AVG(?aTimeSum) as ?aTime_avg) (MIN(?aTimeSum) as ?aTime_min)
       (MAX(?aTimeSum) as ?aTime_max)
WHERE{
  SELECT ?a ?ai (SUM(?aTime) as ?aTimeSum)
  WHERE{
       SELECT ?a ?ai ((?e2Time-?e1Time) as ?aTime)
       WHERE{
        ?e1 event:originatedFromActivityInstance ?ai; a event:RunningEvent.
        ?e2 event:originatedFromActivityInstance ?ai.
        ?e2 event:causedBy ?e1.
        ?e2 event:hasTimeMilli ?e2Time.
        ?e1 event:hasTimeMilli ?e1Time.
        ?ai event:definedByActivity ?a.
        ?a workflow:hasActivityName "Analyze Case".
        ?a workflow:partOfWorkflow ?w.
        ?w workflow:hasWorkflowName "Complaint Handling".
       }
  }
  GROUP BY ?a ?ai
}
GROUP BY ?a
```

?aTime_avg	?aTime_min	?aTime_max
21000.0	20000	22000

Query 4 Answering Competency Question 3 (Running Time)

The result shows that activity *Analyze Case* was, on average, processed for 21000 milliseconds, whereas the shortest processing time amounts 20000 and the longest 22000 milliseconds. To calculate the suspend time of the same activity, only the first two lines of the most inner WHERE-statement have to be exchanged.

```
…
?e1 event:originatedFromActivityInstance ?ai; a event:SuspendedEvent.
?e2 event:originatedFromActivityInstance ?ai; a event:RunningEvent.
…
```

?sTime_avg	?sTime_min	?sTime_max
12500.0	10000	15000

Query 5 Answering Competency Question 3 (Suspend Time)

SPARQL queries can also be used to answer questions about the control flow. The following query answers the questions of how many different execution paths were traversed for workflow “Complaint Handling” and how often they were traversed each. The most inner WHERE-statement collects for each activity instance belonging to a workflow instance the first event. From these first events, the order of the executed

activities is reconstructed and concatenated to a string of activities. The most outer query counts the number of execution paths.

```
SELECT ?excecutionOrder (COUNT(?excecutionOrder) as ?number)
WHERE
{
  SELECT ?wi (GROUP_CONCAT(?aName; separator = ',') as ?excecutionOrder)
  WHERE{
    SELECT ?wi ?ai ?aName (MIN(?time) as ?startTime)
    WHERE{
     ?event event:originatedFromActivityInstance ?ai.
     ?event event:hasTimeMilli ?time.
     ?ai event:definedByActivity ?a.
     ?a workflow:hasActivityName ?aName.
     ?ai event:partOfWorkflowInstance ?wi.
     ?ai event:definedByActivity ?a.
     ?wi event:definedByWorkflow ?w.
     ?w workflow:hasWorkflowName "Complaint Handling".
    }
    GROUP BY ?wi ?ai ?aName
    ORDER BY ?wi ?ai ?startTime
  }
  GROUP BY ?wi
}
GROUP BY ?excecutionOrder
ORDER BY ?number
```

? executionOrder	?number
Create New Case,Analyze Case,Notify Customer,File Case	1
Create New Case,Analyze Case,Provide Solution,File Case	1

Query 6 Answering Competency Question 4

The results show that the workflow was executed twice in total while each time a different path was traversed.

To answer how much time a certain user spent on his assigned activities, the query for calculating the running time can be reused. This time, however, activities executed by a certain user are filtered (see next page).

```
SELECT ?pName ?wName ?aName (SUM(?aTimeSum) as ?aTimeTotal)
WHERE{
  SELECT ?pName ?w ?wName ?a ?aName ?ai (SUM(?aTime) as ?aTimeSum)
  WHERE{
       SELECT ?pName ?w ?wName ?a ?aName ?ai ((?e2Time-?e1Time) as ?aTime)
       WHERE{
        ?e1 event:originatedFromActivityInstance ?ai; a event:RunningEvent.
        ?e2 event:originatedFromActivityInstance ?ai.
        ?e2 event:causedBy ?e1.
        ?e2 event:hasTimeMilli ?e2Time; event:producedByAccount ?p.
        ?e1 event:hasTimeMilli ?e1Time; event:producedByAccount ?p.
        ?p system:hasAccountUserName ?pName.
        ?ai event:definedByActivity ?a.
        ?a workflow:hasActivityName ?aName.
        ?a workflow:partOfWorkflow ?w.
        ?w workflow:hasWorkflowName ?wName.
        FILTER (?pName = "analyst1@domain.de").
       }
  }
  GROUP BY  ?pName ?w ?wName ?a ?aName ?ai
}
GROUP BY ?pName ?w ?wName ?a ?aName
ORDER BY DESC(?aTimeTotal)
```

?wName	?aName	?number
Complaint Handling	Analyze Case	42000
Complaint Handling	Provide Solution	20000
Complaint Handling	Notify Customer	20000

Query 7 Answering Competency Question 5

It can be seen that the user *analyst1@domain.de* worked the most on activity *Analyze Case*, namely in total for 42000 milliseconds.

A modified version of the previous query can also be used to calculate the overall active times for all roles giving an indication how work is divided onto the different roles.

```
SELECT ?p ?pName (SUM(?aTimeSum) as ?aTimeTotal)
WHERE{
  SELECT ?p ?pName ?ai (SUM(?aTime) as ?aTimeSum)
  WHERE{
       SELECT ?p ?pName ?ai ((?e2Time-?e1Time) as ?aTime)
       WHERE{
        ?e1 event:originatedFromActivityInstance ?ai; a event:RunningEvent.
        ?e2 event:originatedFromActivityInstance ?ai.
        ?e2 event:causedBy ?e1.
        ?e2 event:hasTimeMilli ?e2Time; event:producedByAccount ?acc.
        ?e1 event:hasTimeMilli ?e1Time; event:producedByAccount ?acc.
        ?acc event:belongsToParticipant ?p.
        ?ai event:definedByActivity ?a.
        ?p a workflow:RoleParticipant; workflow:hasParticipantName ?pName.
       }
  }
  GROUP BY ?p ?pName ?ai
}
GROUP BY ?p ?pName
```

?wName	?aName	?number
http://www.complaint.com/model.rdf#Participant1	ServiceAgent	40000
http://www.complaint.com/model.rdf#Participant2	Analyst	82000

Query 8 Answering Competency Question 6

The result shows that users assigned to the role *Analyst* have, in total, about twice as much running time than users assigned to role *ServiceAgent*.

When reasoning is applied to the created data, the reasoner detects (using the axioms defined in the ontology) which business partners represent the same real world entity. It is done by automatically comparing the first name, last name and email address and creating an *owl:same as* relationship between those business partners where the properties are the same. This new relationship induces new inferences: Every relationship of the first business partner that the second business partner does not have is copied to the second one (and vice versa). Through this mechanism, every workflow instance is eventually connected to every business partner that took part in the instance even though the business partner might have been defined initially inside another system. The same principle is used to detect if different economic objects describe the same object. The SPARQL query,

presented in the following, exploits this fact and queries all customers that first ordered a product and then complained about the same product.

```
SELECT DISTINCT ?fName ?lName ?eMail
WHERE{
  ?bp a businesspartner:Customer;
        businesspartner:hasFirstName ?fName;
        businesspartner:hasLastName ?lName;
        businesspartner:hasEmailAddress ?eMail.
  ?eo a economicobject:EconomicObject; economicobject:hasOrderReferenceNumber ?orn.
  ?wi1 event:hasBusinessPartnerContext ?bp; event:hasEconomicObjectContext ?eo.
  ?wi2 event:hasBusinessPartnerContext ?bp; event:hasEconomicObjectContext ?eo.
  ?wi1 event:definedByWorkflow ?w1.
  ?wi2 event:definedByWorkflow ?w2.
  ?w1 workflow:hasWorkflowName ?w1Name.
  ?w2 workflow:hasWorkflowName ?w2Name.
  FILTER (?w1Name = "New Order" && ?w2Name = "Complaint Handling")
}
ORDER BY ?fName ?lName ?eMail
```

?fName	?lName	?eMail
Adam	Williams	adam.williams@domain.com

Query 9 Answering Competency Question 7

The result shows that Adam Williams is the only customer that first ordered a product and then complained about it.

6 Prototype Development

Based on the ontology presented in chapter 4, a prototype for importing and analyzing process event data was developed. The development comprised the definition of functional requirements (section 6.1), the conceptualization (section 6.2) as well as the implementation (section 6.3) of the prototype. The prototype was named Semantic Process Analyzer and is referred to as SPA in the following.

6.1 Functional Requirements

The general purpose of the prototype is to provide an intuitive interface enabling the analysis of arbitrary audit trail event data originating from different sources. In detail, the following requirements served as a basis for the prototype development.

Usability: The semantic web and its underlying technologies are complex and difficult to understand for people without an IT-background. The prototype should be usable by IT and business experts and should hide as much as possible of the underlying semantic technologies and their inherent complexity.

Availability: The prototype should be easy to deploy and available from a variety of different operating systems. Hence the prototype should be developed in an operating system independent programming language and be usable via a web-based interface.

Technology: Existing prototypes in this area are either outdated or only show a very limited range of functionality mainly intended for researchers. This prototype should only be based on state of the art semantic web as well as web-development (interface) technologies making it accessible to a broad range of different users. To be consistent, every kind of data that has to be saved (including user accounts) should be saved using semantic web technologies.

Import: The prototype should be generally capable of importing audit trail event data from arbitrary BPMS. Hence, it should be developed in a generic manner that allows for the extension of the existing functionality and for the development of new import components. At least two different BPMS should be supported by the prototype.

In addition to the import of real-life event data, a component for the import of synthetic event data should be available allowing for an easy demonstration of the provided functionality in case suitable BPMS are not available. Reasoning over imported data should be manually controllable. To simplify the import, different workflow versions will be neglected and only the newest version will be imported.

Analysis: The prototype should at least provide the functionality to answer the competency questions introduced in the ontology development chapter (chapter 4). Beyond that, the definition and execution of custom SPARQL queries should be supported whereas functionality for storing and opening such queries should be provided.

Export: The prototype has to provide an export component that enables a user to export created RDF-data for external tools or further analysis (e.g. verification of created data). The common serialization formats used by the semantic web community such as RDF/XML, RDF/JSON and RDF/Turtle should be supported.

6.2 Conceptualization

During the conceptualization phase, an appropriate semantic application framework for the business logic (section 6.2.1), a web framework for creating the interface (section 6.2.2) as well as two different BPMS for demonstrating the event import (section 6.2.3) were chosen. Based on the technology choices, an architecture for the prototype was developed (section 6.2.5).

6.2.1 Semantic Application Framework

A framework can be understood as a software environment designed to support future development of applications by providing reusable building blocks that can be used, extended or customized to the specific need of a developer. Especially when developing Semantic Web applications, the use of an existing framework greatly facilitates the development process. This is mainly due to the fact that the Semantic Web consists of a multitude of different technologies and concepts (as shown in the foundations chapter) that can make development projects complex and error prone (Yu 2011, p. 467 ff.). In the following, popular Semantic Web frameworks will be presented whereas a special focus lies on Apache Jena which was chosen as the development framework for the prototype.

The main features of most available semantic web frameworks are

- Core support for creating RDF, RDFS and OWL documents,
- Inference capabilities (built-in or though the integration of external reasoners),
- Support for SPARQL queries,
- Handling of persistent RDF models with the ability to scale efficiently to large data sets (using triples stores).

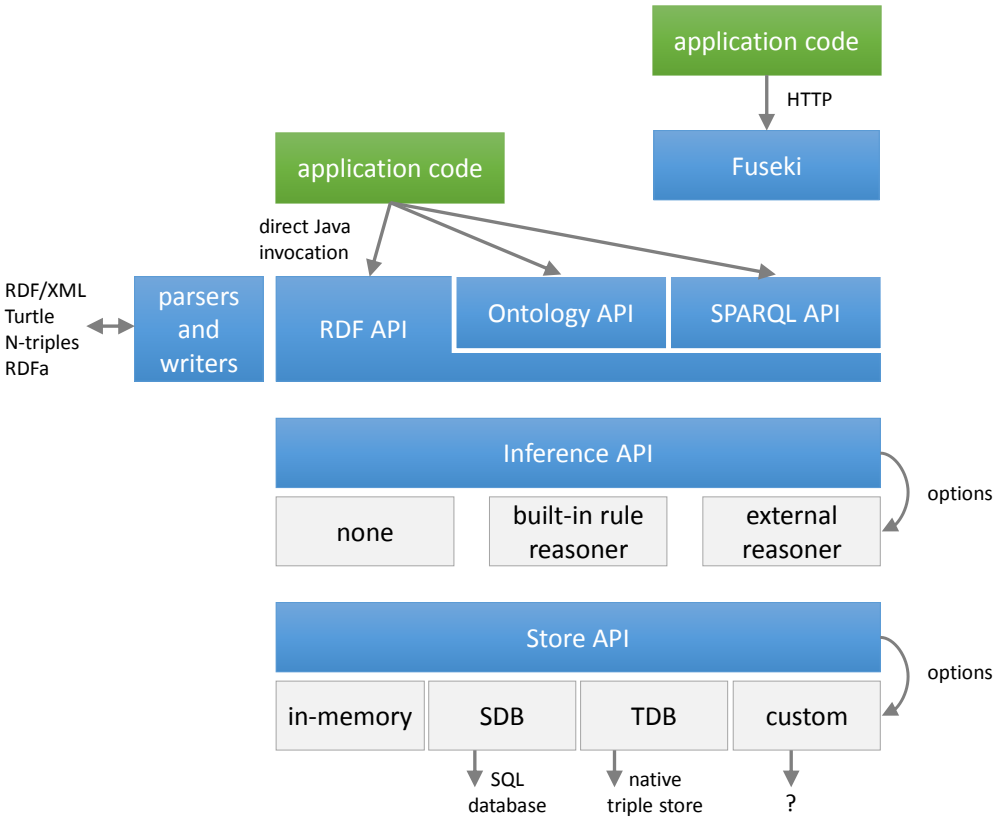
Apache Jena

Jena[12] is a free, open-source Java framework for developing Semantic Web applications. It was originally developed by Brian McBride from Hewlett-Packard Laboratories (HLP). Jena 1 was originally released in 2001, Jena 2 in 2003 and its current version 2.11.2 in June 2014. Jena is one of the most used framework for building Semantic Web applications (Yu 2011, p. 468) and referenced in many academic papers and conferences (e.g. Carroll et al. 2004; Ferreira et al. 2012; Meditskos and Bassiliades 2010; Sunkle et al. 2013). Jena supports all relevant areas of Semantic Web applications and provides the following components:

- RDF API for the import, creation and serialization of RDF models.
- OWL API for importing and creating ontologies.
- In-memory and persistent storage of RDF models (native triple store or relational database).
- SPARQL query engine for querying RDF models.
- SPARQL endpoint (called Fuseki) for publishing RDF data on the web.
- Built in reasoner and support for external reasoners.

In general, Jena provides two possibilities to create and modify RDF models. It is, on the one hand, possible to directly interact with the RDF, OWL and SPARQL API via Java code or, on the other hand, to set up a Jena Fuseki SPARQL endpoint that receives SPARQL queries (e.g. select, insert, delete) via HTTP requests and forwards them to the SPARQL API. For the prototype, the first possibility was chosen since created event data should not be published on the web but solely be available to the application operating on the event data. A subsequent publication via Fuseki, however, is still possible since it can easily be connected to the produced RDF data. Figure 6-1 depicts the different APIs provided by Jena.

[12] Available at: https://jena.apache.org/index.html.



Source: Jena (2014)

Figure 6-1 Jena Application Architecture

Other Frameworks

Sesame[13] is an open-source Java framework for parsing, storing, inferencing and querying RDF data. Sesame was originally developed as a research prototype for an EU research project called On-To-Knowledge and is currently developed as a community project. Its latest version 2.7.12 was released in June 2014. Sesame's focus lies on the RDF instance data without much support for ontologies.

Virtuoso[14], also called Virtuoso Universal Server is actually a database engine that combines the functionality of traditional relational databases with semantic web technologies. It can be used as an RDF Triple store and provides functions to load N3, Turtle and RDF/XML serializations into the store as well as functions for reasoning based on RDFS and OWL. Additionally, SPARQL queries are supported whereas SPARQL can be even integrated into normal SQL queries. This makes Virtuoso very useful when data from semantic and non-semantic data sources should be accessed. Virtuoso includes APIs supporting Jena, Sesame and other semantic web frameworks.

[13] Available at: http://www.openrdf.org/.
[14] Available at: http://virtuoso.openlinksw.com/.

Redland[15] is a set of free C libraries that provide support for parsing and serializing RDF documents (Raptor RDF Syntax Library), executing SPARQL queries (Rasqal RDF Query Library) and storing RDF data in triple stores (Redland RDF Library). The RDF packages are developed independently since 2000. The latest versions of the different libraries are from 2014 (Raptor RDF Syntax Library, Rasqal RDF Query Library) and 2013 (Redland RDF Library).

6.2.2 Web Application Framework and IDE

In order to be able to integrate all functionalities (i.e. Jena libraries) into one single web application, the Java-based web development framework Play[16] was chosen. Play, first released in 2007, follows the so called "Convention over Configuration" design paradigm which means that complex configuration is avoided as far as possible to facilitate the development workflow and increase manageability of the web application's code (Bächle and Kirchberg 2007). Although based on Java, Play was specifically designed for web development and focuses on a support of the main web development technologies like HTML, CSS and JavaScript. This distinguishes it from other popular Java-based frameworks like JavaEE which have a stronger focus on Java rather than the web development standards and provide more flexibility (Play 2014a). Play is scalable and can be used for arbitrarily sized projects. It is also used by LinkedIn and the Guardian (Play 2014b). As an integrated development environment (IDE), IntelliJ 13 was chosen which is one of the recommended IDEs for developing with Play (Play 2014c).

6.2.3 BPMS

To be able to demonstrate the import of real-life event data as well as its transformation into an ontological representation, the market for open source BPMS was evaluated and two different systems with two different internal state models were chosen. The first system is Eclipse Process Manager (also called Stardust) and the second one is Camunda.

Stardust

The origin of Stardust[17] is the Java-based BPMS CARNOT developed by the German start up CARNOT AG in Frankfurt, founded in 2000. In 2006, CARNOT was acquired by SunGard's infinity technology initiative and renamed Infinity Process Platform. In 2007, the process platform was ranked second in Gartner's Magic Quadrant for BPMS and used throughout the industry. SunGard's strategy for Infinity includes leveraging

[15] Available at: http://librdf.org/.
[16] Available at: http://www.playframework.com/.
[17] Available at: http://wiki.eclipse.org/Stardust/Download.

open source initiatives and contributing back to the open source community. As part of this strategy, SunGard decided to make the Infinity Process Platform available as open source via Eclipse which has been the technological foundation since 2004. The product, now called Eclipse Process Manager (Stardust), is freely available as an Eclipse plugin. According to the developers, the Stardust contribution is one of the largest ever made to Eclipse. Stardust includes components for process modeling, process execution, simulation, monitoring and reporting. Process models defined in stardust are saved in the XPDL format (Stardust 2013, 2014a).

Camunda

Camunda BPM[18] is a light-weight, open source platform for Business Process Management. It is, like Stardust, Java-based and fully integrated into Eclipse. Camunda BPM is an originally commercial system of the Berlin-based company Camunda which is based on Activiti, another open source BPMS. In March 2013, Camunda changed its strategy and made their product available as open source. Besides the open source version, the company still offers an enterprise version with extended functionality (e.g. extended event state model). Camunda is used by companies such as Allianz or Zalando AG. In contrast to Stardust, Camunda is fully based on the BPMN 2.0 standard and its BPMN XML serialization format for process models (Camunda 2014; Signavio 2013; Wikipedia 2014b). In contrast to Stardust, Camunda does not provide a simulation function and only limited reporting capabilities (see section 6.3.2).

6.2.4 SPA Ontology

As described in the functional requirements, all kinds of data that have to be saved by the SPA should be saved using semantic technology as well. Necessary data to be saved are SPA user accounts and queries saved by users.

6.2.5 Architecture

Based on the technology choices, the following application architecture was conceptualized and used for the further development process. The final architecture is the result of several refinements during the development phase based on increased experiences with the two frameworks (see Figure 6-2). All functionalities are embedded into one single Play-project. The Play framework supports the popular Model-View-Controller pattern which separates the interface (view) from the business logic (controller) which itself is separated from the persistence (model). The interface consists of several distinct views providing the necessary interaction logic for importing event

[18] Available at: http://camunda.com/.

data from several different sources (synthetic or BPMS), to generate serializations of the created RDF data (export), to display an overview/summary of the existent event data, to generate statistics (e.g. time metrics) and to execute custom SPARQL queries against the RDF data. Every view makes use of web services defined in Play that execute certain parts of the business logic and deliver a result that can be displayed in the view.

The business logic covers three tasks: model creation (including import of event data), reasoning and the execution of SPARQL queries. To facilitate model creation and provide a unified API for the creation of RDF models for storing event data, a separate *SPI Model API* was created on top of the *Jena RDF API* and *Jena OWL API*. One task of the *SPI Model API* is, for instance, to load and verify the event ontology files published on GitHub. It also provides methods for creating all necessary objects such as activities, workflows or events without having to deal with the underlying resource URIs for each object. The API is then used by different event importers to create an actual model. For this prototype, a synthetic importer for importing artificial event data provided in a proprietary format, as well as two different importers for the import of real life event data from two different BPMS were developed (see section 6.3.2). Once the import and model creation is finished, the web service that initiated the process, can reason over the created model using the Jena Inference API and save the model into the triple store using the *SPI Persistence API* built on top of the *Jena Store API*.

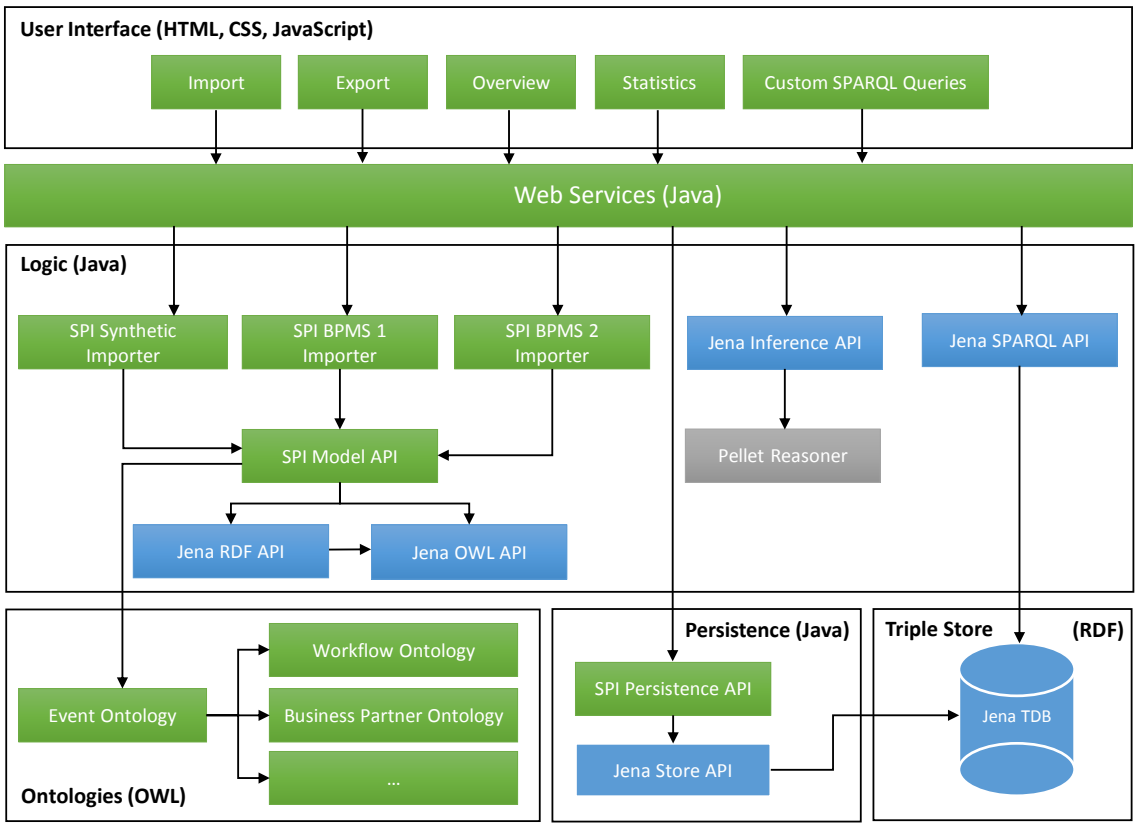


Figure 6-2 Prototype Architecture

For reasoning, the external Pellet[19] reasoner was embedded since it turned out that the internal reasoner was not powerful enough to e.g. infer *owl:same as* properties as needed for identifying equivalent real world business partners and economic objects. Pellet is a fully functional OWL reasoner supporting the newest OWL version (2.0) recommended by the W3C. It is used regularly in literature (e.g. Horridge et al. 2009; Vysniauskas et al. 2010) and, according to the developers' website, one of the leading choices for reasoning. The SPI Persistence API provides functions for simplifying the extraction of RDF data from the triple store as well as for saving RDF data into it. The Jena SPARQL API directly operates on the triple store and is used (via web services) by the overview, statistics and customer SPARQL query view.

For data persistence, the Jena Triple Database (TDB) was chosen since the integration with Jena is simple and the performance, also for a large number of triples, reasonable. The integration of third party triple stores, however, is relatively easy due to a variety of additional APIs. Jena also supports traditional relational databases whereby a simple schema for saving all triples in one table is used. Native triple stores however provide a better performance since queries don't have to be translated into SQL queries first and they are optimized for storing and querying triples (Bizer and Schultz 2008).

6.3 Implementation

In the following, the most important SPA components will be presented. These are the Model API (section 6.3.1) which is used by every importer to create and connect event data, the three different importers to import event data from different sources (section 6.3.2), the SPA ontology for saving internal data (section 6.3.3) as well as the user interface (section 6.3.4). More information about the IntelliJ project structure as well as a deployment tutorial can be found in appendix F and G. Code statistics are listed in appendix D.

6.3.1 Model API

As described in section 6.2.5, the Model API was developed to facilitate the creation of RDF data based on the event ontology. Although Jena already provides methods that support the creation of RDF data, much additional and redundant code has to be written to create resources, to assign resources to ontology classes, to create object properties to connect different resources as well as to create data properties (to name a few tasks). This is due to the flexibility of the Jena framework which was not developed with a specific ontology in mind. The Model API provides methods to create every resources type (class)

[19] Available at: http://clarkparsia.com/pellet/.

defined in the ontology as well as to connect the resources as prescribed in the ontology. The Model API additionally manages all created resources in lists of key-value-pairs such that an already created resource can be easily retrieved by providing its identifier. This facilitates the connection of already created resources. By hiding the complexity in a separate API, the importer classes require less code and are less error-prone.

6.3.2 Import

Every import requires the same set of tasks: (1) import and reconstruction of the workflow model, (2) import and creation of event data (including addition of connections to the workflow model) and (3) import and creation of context data (including addition of connections to the workflow instances). This motivated the creation of an abstract class *Importer* that prescribes necessary methods and contains redundant code. Every concrete implementation of an importer extends this abstract class and adds its system specific implementation to the defined methods (see Figure 6-3). Since the creation of event and context data cannot always be separated, only two abstract methods are prescribed.

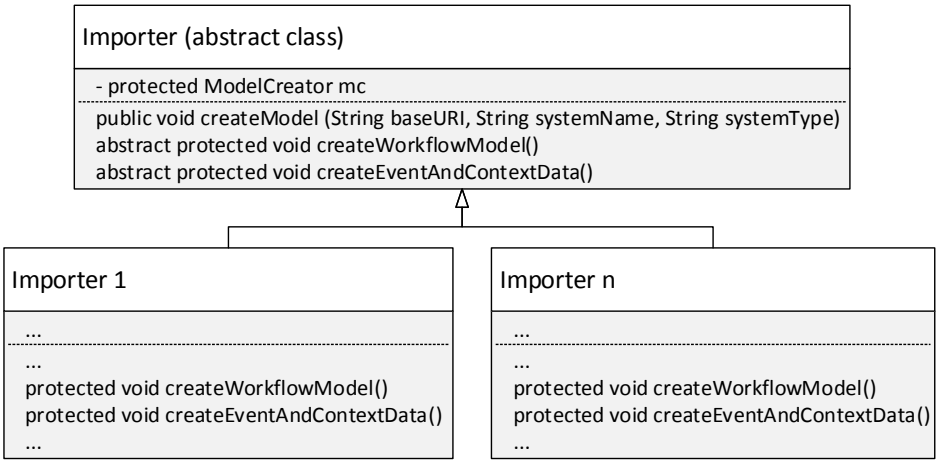


Figure 6-3 Importer Classes

Three importers were developed. In the beginning, an importer for synthetic event data based on a proprietary format was developed. The motivation behind this importer was that in the beginning it wasn't clear if the chosen BPMS would provide all necessary event data to show the full potential of the developed ontology. Subsequently, the importer classes for the chosen BPMS Stardust and Camunda were developed.

Synthetic Importer

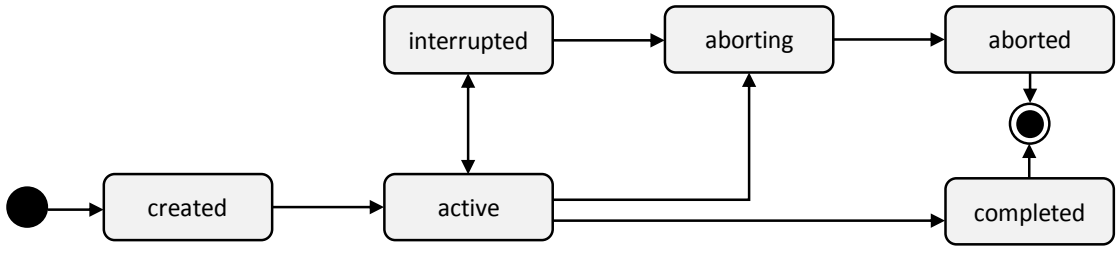
It was decided to build the synthetic importer based on Excel files. One reason for this decision was that Excel files are easy to import and another (more important one) that Excel files are easy to create, also by less BPI experienced persons.

The synthetic importer imports event data from three different Excel files. The first file (workflow.xlsx) describes the workflow model according to the XPDL specification whereby participants, applications, workflows, activities, transitions and control flows (XOR/AND) have to be described in a tabular form. The definition of packages was excluded from this file since it wasn't considered necessary for the analysis. The second file (event.xlsx) describes the event data whereby workflow instances, activity instances, workflow events and activity events have to be created using the Ids defined in the workflow document. The third and last file (context.xlsx) describes the context of the defined workflow instances whereby business partners and economic objects have to be described. Wherever necessary, possible types (e.g. participant, activity or event type) are prescribed by dropdown fields in the respective document so that it is consistent with the types defined in the ontology and can be easily translated to an RDF document. The Excel files used for the synthetic *Complaint Handling* workflow can be found in appendix E.

Stardust Importer

Stardust provides two methods to for accessing saved event data. The first and official method is to use the provided web services (both SOAP and REST are available) offered by the engine. The result of an initial test prototype, however, was that due to insufficient documentation and examples, the retrieval of all necessary information could not be accomplished. It was therefore decided to directly access the underlying audit trail database (which is also remotely accessible via the network) and to manually extract the needed information.

The next step in creating the Stardust importer was to map the internal event state model to the model defined in the ontology. Stardust provides two different state models for workflows and activities as depicted in Figure 6-4 and Figure 6-5. A workflow instance execution begins in the state *created* and transitions to *active* once a user or application has started workflow on it. From here on, the instance can be *interrupted* (suspended) or *aborted*. *Interrupted* means that one of the activities in the workflow caused an exception. An abortion always leads to the state *completed* (Stardust 2014b).



Source: Stardust (2014c)

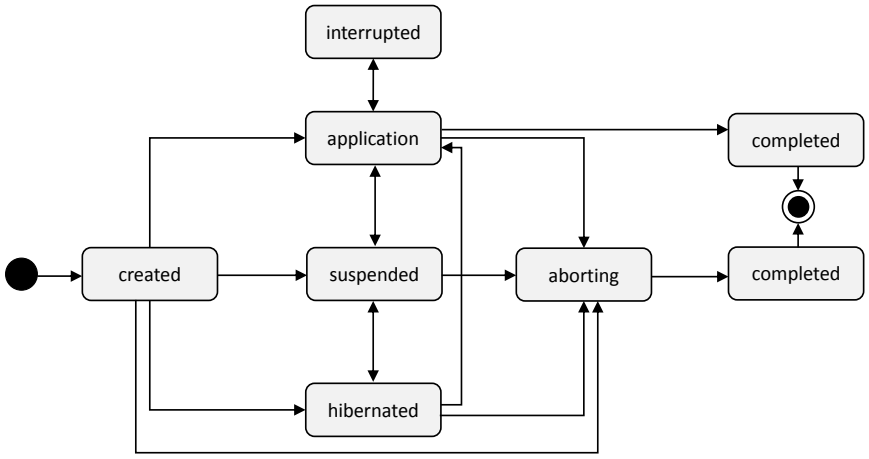
Figure 6-4 Stardust: Workflow Instance States

The Stardust workflow instance states were mapped the following to the ontology states:

Stardust State	Ontology State
created	Open.NotRunning.Ready
interrupted	Open.NotRunning
active	Open.Running
aborting	Not necessary due to state *aborted*
aborted	Closed.Cancelled.Aborted
completed	Closed.Completed

Table 6-1 Mapping of Stardust Workflow Instance States to Ontology States

The state model for activity instances applied by Stardust is similar to the one for workflow instances but has two additional states *suspended* and *hibernated*. While the meaning of *suspended* is obvious, hibernated refers to a state in which an activity is active but waiting to be triggered by an incoming event/message. This can for instance be an email or a JMS (Java Messaging Service)-message. Stardust provides several trigger types that can be connected to activities for that purpose (Stardust 2014b).



Source: Stardust (2014d)

Figure 6-5 Stardust: Activity State Model

The Stardust activity instance states were mapped the following to the ontology states:

Stardust State	Ontology State
created	Open.NotRunning.Ready
application	Open.Running
suspended	Open.NotRunning.Suspended
hibernated	Open.NotRunning.Ready
interrupted	Open.NotRunning
aborting	Not necessary due to state *aborted*
aborted	Closed.Cancelled.Aborted
completed	Closed.Completed

Table 6-2 Mapping of Stardust Activity Instance State to Ontology States

The hibernated state was mapped to *Open.NotRunning.Ready* since the activity is in a waiting state and has not started its execution yet.

Stardust saves all data values that are used by activities as input or created as output during the execution of a workflow instance to the audit trail database. To be able to associate created data values with the classes and data properties defined in the ontologies, the following mechanism was applied: From the available data types only the structured data type was used. In Stardust, a structured data type object is an object that has a name and several fields associated to it (internally represented using XML). The object's name was used to provide information about the ontology class. For example the name BusinessPartner#Person_Customer means that the objects describes properties of a business partner (business partner ontology) that is of type *Person* and of type *Customer*. All fields of the object are named using the data property names of the respective ontology. For instance *FirstName*, *LastName* and *EmailAddress* (whereby the verb *has*, which is used in the ontology, was left out). Using this notation, all data saved in the audit trail database can be transferred to an ontological representation.

Stardust's audit trail database has some limitations. Most elements of the model defined with the Stardust modeler are available through the audit trail database. Defined control flows (XOR, AND) and applications connected to the activities however are not available and could hence not be imported.

Camunda Importer

In contrast to Stardust, Camunda provides a comprehensive and well documented web service for accessing event data[20]. The current version (7.1.0), however, has one limitation: It only focusses on providing information about executed workflows and activities but not about those that haven't been executed. This makes it difficult to reconstruct the workflow model. To circumvent this limitation, the BPMN XML file for every workflow definition had to be retrieved (available via the web service) and missing information be extracted manually. For parsing the file, the publicly available Camunda BPMN parser was used which is also used as part of the Camunda BPM suite.

Camunda BPM supports common functions like assigning tasks to other users or to suspend the work on a current task. In its current version, the Camunda web service however does not provide a function for retrieving an overview of all such events. Only the start and end times of workflows and activities can be retrieved. While this fact limits the possibilities for further analysis by the prototype, it can be used to show how the prototype copes with different levels of event granularity. In contrast to Stardust,

[20] Camunda Web Service Overview: http://docs.camunda.org/7.1/api-references/rest/#overview.

Camunda only saves timestamps with a precision of seconds which makes it for instance difficult to analyze the execution order of automatic activities that are executed in less than a second.

Similar to Stardust, Camunda can save (context) data that were entered or created during the execution of a workflow to the audit trail database. A common method to provide activities with input data is by specifying HTML-forms that are assigned to the respective activity. During the execution of the activity, Camunda recognizes the input values and saves them under the field name specified in the form into the database. The web service provides a method for retrieving this data. To be able to recognize the semantic meaning of the data, the following naming convention was applied to all form fields: OntologyName#ClassName1_ClassNameN#DataPropertyName. Since the web service did not provide any information about which data values belong to the same form and hence same resource (e.g. Customer), the following rule was applied: All data values that have the same ontology and class names and belong to the same workflow instance are assumed to belong to the same resource.

Camunda uses a different participant model. It includes *assignee, candidate users* and *candidate groups*. *Assignee* was mapped to *HumanParticipant*, and *candidate group* to *RoleParticipant*. *Candidate users* was not imported.

6.3.3 SPA Ontology

The SPA ontology is based on the system ontology presented in section 4.4.4 as well as a newly developed simple ontology for saving queries.

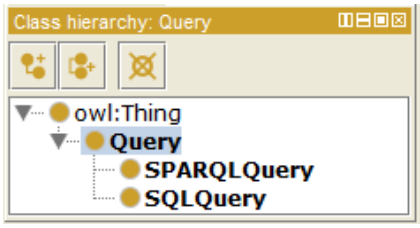


Figure 6-6 Query Ontology: Classes

The Query ontology only contains the following data properties.

Data Property	Domain	Range
hasQueryName	Query	String
hasQueryDescription	Query	String
hasQueryContent	Query	String

Table 6-3 Query Ontology: Data Properties

The SPA ontology imports the query and the system ontology and adds a new object property *createdByAccount* connecting queries with accounts.

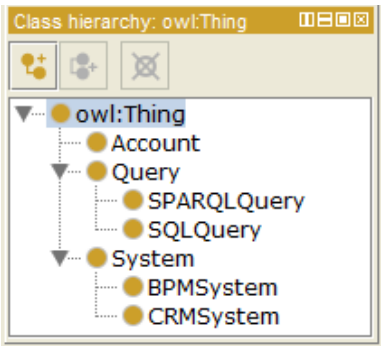


Figure 6-7 SPA Ontology: Classes

6.3.4 User Interface

The user interface is completely web-based and was solely developed using HTML, CSS and JavaScript. It communicates with the presented import logic via sending HTTP-Post requests to several java web services defined with the Play framework. By using Post requests, the functionality cannot be accidentally triggered by entering the wrong URL into the Browser address field. To retrieve and display imported data from the triple store, the different web pages send SPARQL queries to an additional web service that executes the query using the Jena SPARQL API and returns the result as a JSON answer. This result is further interpreted and used for generating statistics. As SPARQL queries, those used to answer the competency questions presented in section 5.3 are used and slightly modified. The modifications are in most cases restricted to adding filter statements that are filled with values selected via the user interface.

The SPA covers the five areas *Import*, *Export*, *Summary*, *Statistics* and Custom *SPARQL Query*, whereas the *Statistics* area is further divided into *Quality*, *Time* and *Control Flow* Statistics. The SPA includes a simple user management so that only authorized users can access the application. Two different kinds of users are distinguished: *Admins* and *Analysts*. Admins have access to all areas of the application while analysts have no access to the import and the settings area and can only analyze data. The exemplary users *Ada Admin* (user: ada.admin@test.com, password: test) and *Adam Analyst* (user: adam.analyst@test.com, password: test) are included in the prototype. Further, custom SPARQL queries are saved per user so that only the author of a custom query can later access it via the open query function. The screenshots shown in the following depict the prototype as shown for admins. When logged in as an analyst, the forbidden areas are hidden and not accessible.

The developed user interface is based on the Bootstrap framework[21] which is a set of CSS and JavaScript-files facilitating the creation and design of web pages. The *Custom SPARQL Query* page makes use of the JavaScript framework codemirror[22] for adding SPARQL syntax highlighting to the SPARQL editor. The generated result tables are created using the DataTables[23] plugin for jQuery. All generated charts are created using the Google Visualization API[24], a JavaScript framework for creating interactive charts.

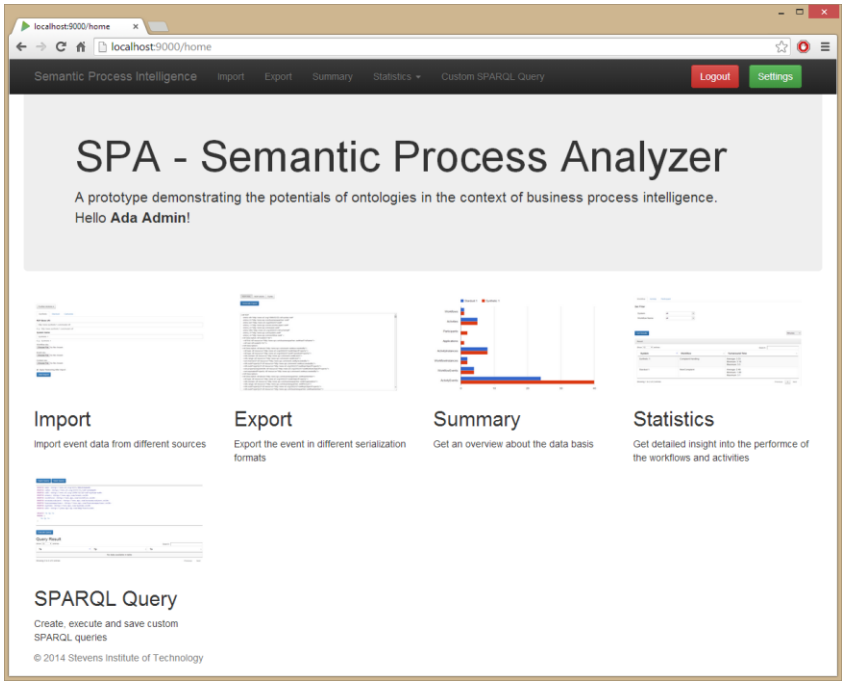


Figure 6-8 SPA – Welcome Page

The import area supports the import of event data from four different sources: *RDF*, *Synthetic*, *Stardust* and *Camunda*. For the RDF import, an RDF file containing the triples in RDF/XML format has to be selected. For the other import types, the RDF base URI (the URI that precedes every resource identifier) as well as the system name have to be specified. For the *Synthetic*, *Stardust* and *Camunda* import, the event sources have to be specified. For the synthetic import, the three excel-files containing the workflow definition (workflow.xlsx), event data (event.xlsx) and context data (context.xlsx) have to be selected.

[21] Download: http://getbootstrap.com/.
[22] Download: http://codemirror.net/.
[23] Download: http://www.datatables.net/.
[24] Overview: https://developers.google.com/chart/interactive/docs/index.

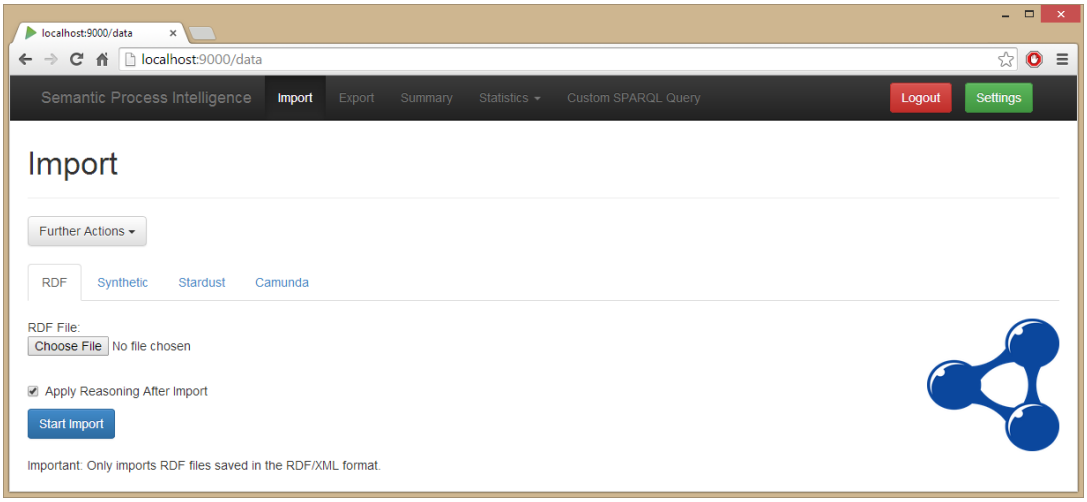


Figure 6-9 SPA – RDF Import

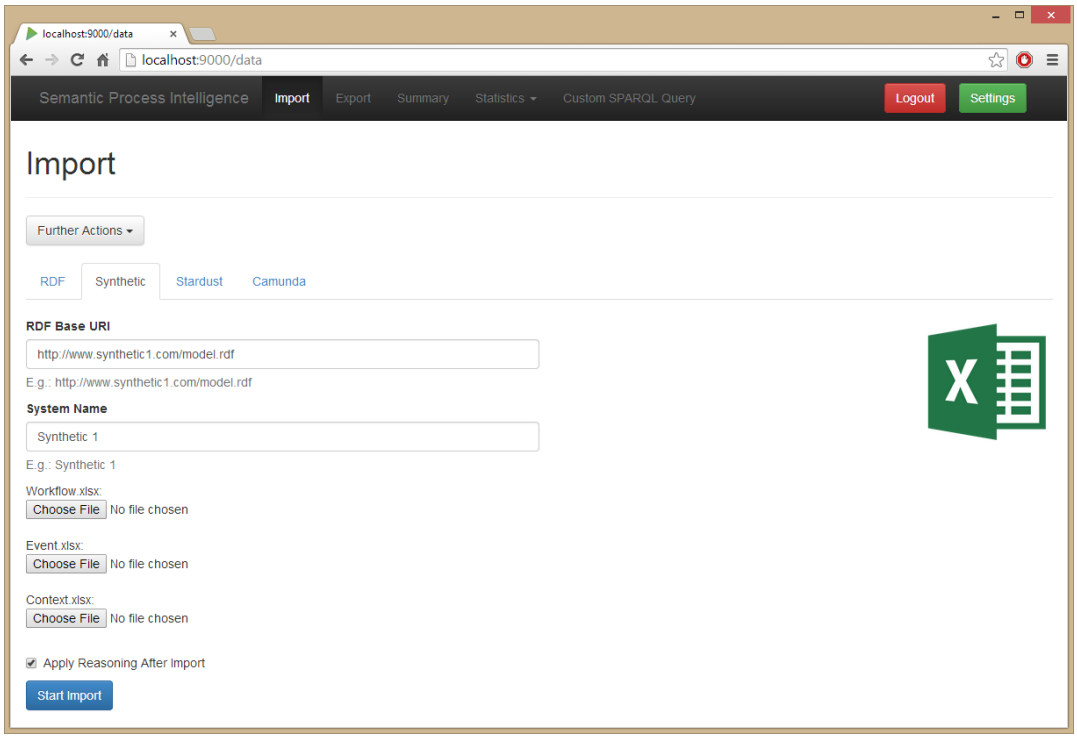


Figure 6-10 SPA – Synthetic Import

For the *Stardust* import (see Figure 6-11), the database URL and for the *Camunda* import (see Figure 6-12), the Web Service URL have to be specified. In each case it is possible to activate or deactivate an automatic reasoning over the imported data. The menu *Further Actions* allows for a manual deletion of all event data contained in the triple store and to manually reason over the existing data. The first option can for instance be used in case automatic reasoning was deactivated for the import. In cases where event data has to be imported from many different sources, deactivating automatic reasoning and applying it at the end accelerates the import process.

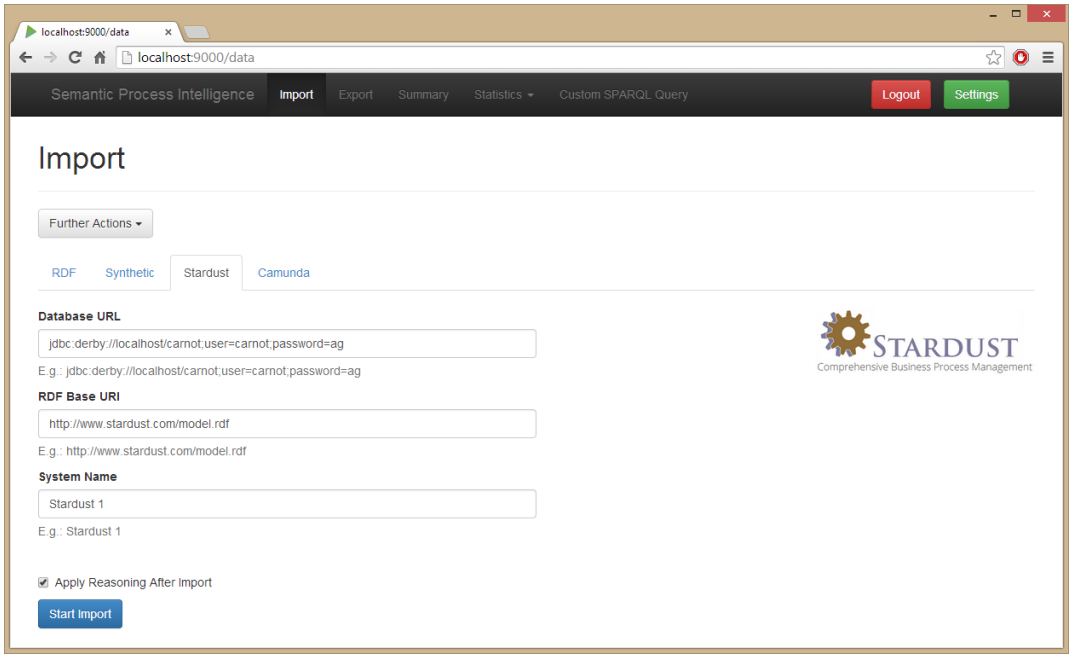


Figure 6-11 SPA – Stardust Import

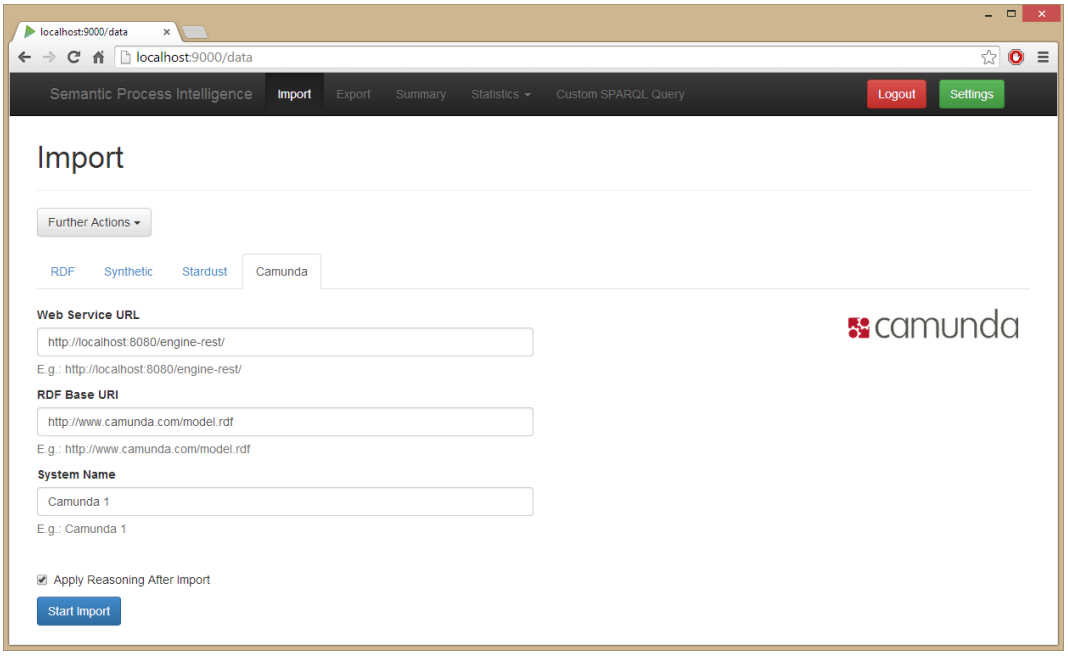


Figure 6-12 SPA – Camunda Import

The export area (see Figure 6-13) allows for serializing the imported event data into the common serialization formats RDF/XML, RDF/JSON and Turtle. From the text area, the generated output can be easily copied or searched via the Browser internal search function.

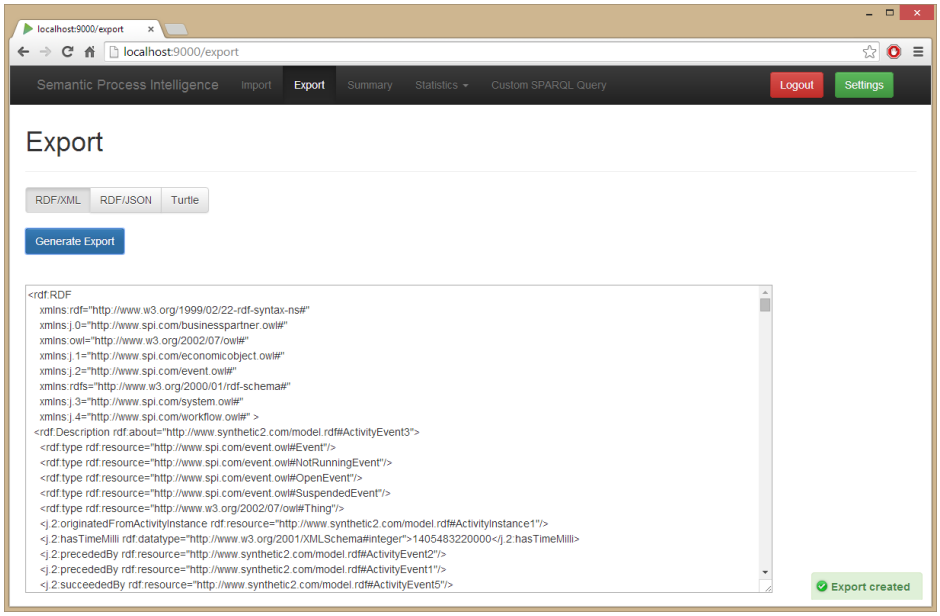


Figure 6-13 SPA – Export

The summary section provides the user with a graphical overview of the imported data whereby the number of imported instances per system is displayed. Figure 6-14 depicts the case where event data from two synthetic sources and a Stardust BPMS were imported.

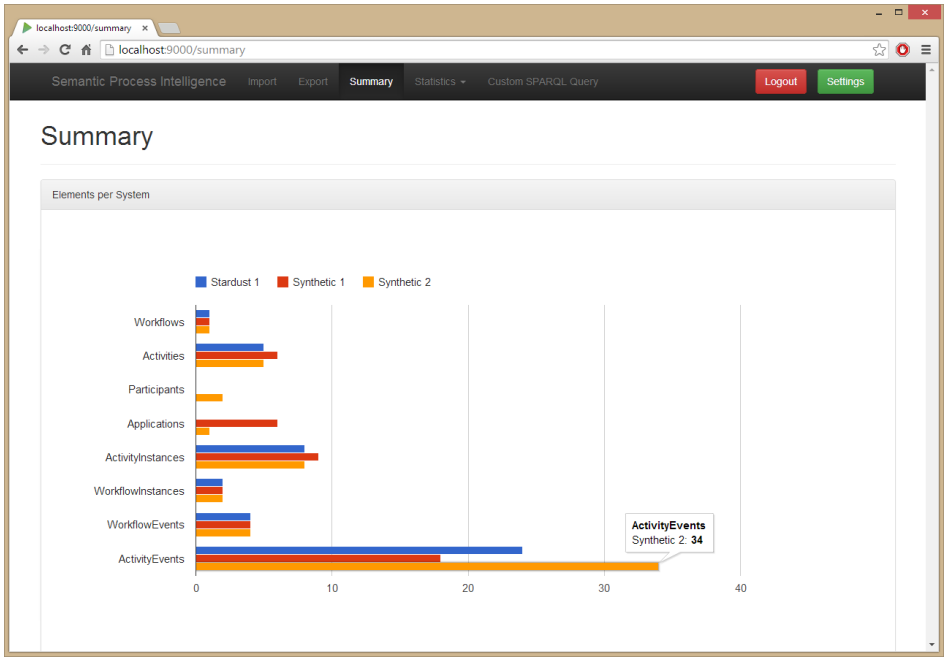


Figure 6-14 SPA – Summary

The statistics area is divided into quality, time and control flow statistics. The quality area provides information about how successfully workflows are executed. At the moment, this area only provides information about the relation between opened and completed workflow instances.

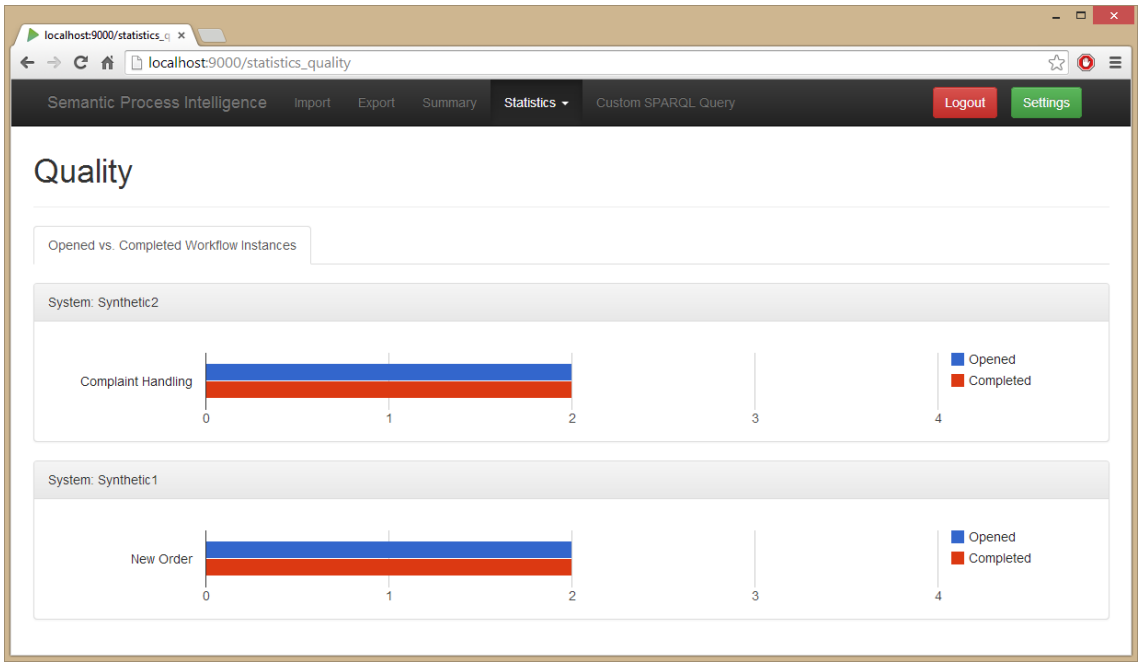


Figure 6-15 SPA – Statistics (Quality)

Time statistics are divided into workflow, activity and participant statistics. Each of these areas have in common that available workflows, activities or participants can be filtered using the provided combo-boxes (combo-boxes allow for searching the available options in contrast to standard dropdown fields). As options, only human friendly names and not URIs are displayed. In the background, however, each name is bound to a unique URI such that the result window does not produce unwanted results. In cases where for instance two activity names from two different workflows are the same, the filter on the next higher level can be changed to only display the desired values or to find out which name comes from which source. In addition to the filter possibilities, each area provides a dropdown field for changing the displayed time format (seconds, minutes, hours and days are available). Further, the result table contains a search field that can be used to dynamically filter the result set.

The workflow area (see Figure 6-16) allows for retrieving statistics about the average, minimum and maximum turnaround time of selected workflows based on the available workflow instances. The metrics are calculated by using the available activity events since the availability of activity events is more reliable than the one of workflow events. The result is presented in a tabular form enabling sorting and search for values using the integrated functionality.

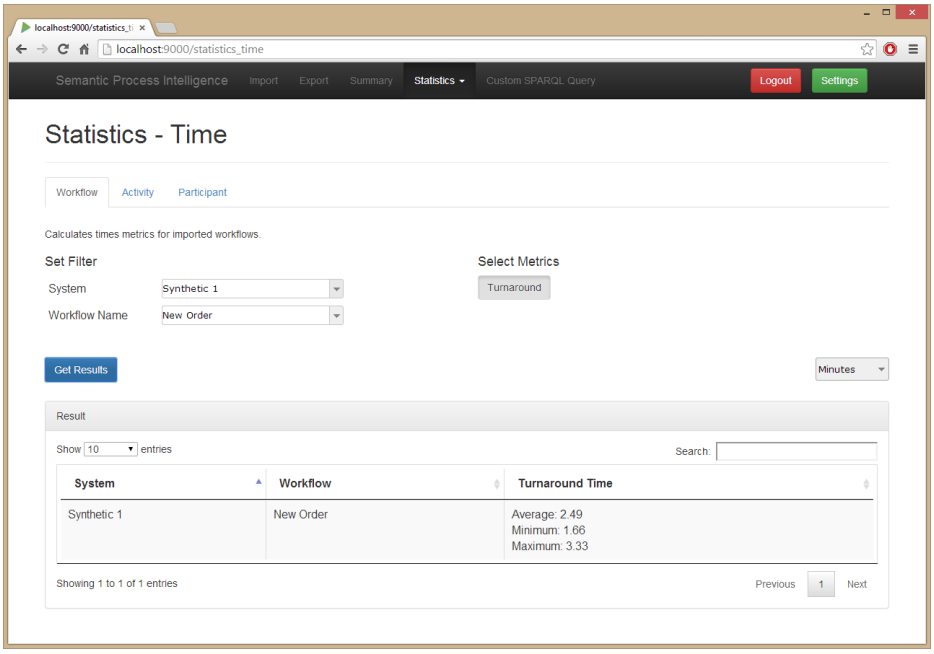


Figure 6-16 SPA – Statistics (Time/Workflow)

The activity section (see Figure 6-17) provides a similar functionality whereas the metrics are calculated for selected activities instead of workflows. The filter therefore provides another selection level. Additionally to the turnaround time, running and suspend time can be calculated. In case a selected metric cannot be calculated, it is indicated in the result with “Not available”.

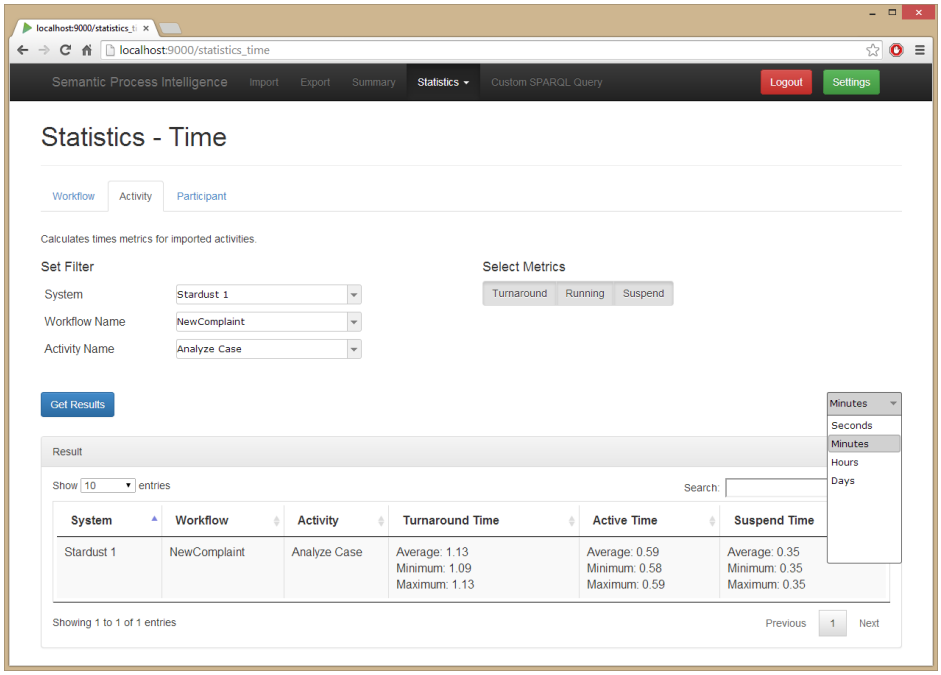


Figure 6-17 SPA – Statistics (Time/Activity)

The participant area (see Figure 6-18) allows for the creation of charts summarizing the total running or suspended time of activities performed by the selected users. The results

are grouped by the selected user while a separate chart is created for each workflow in which the user participated.

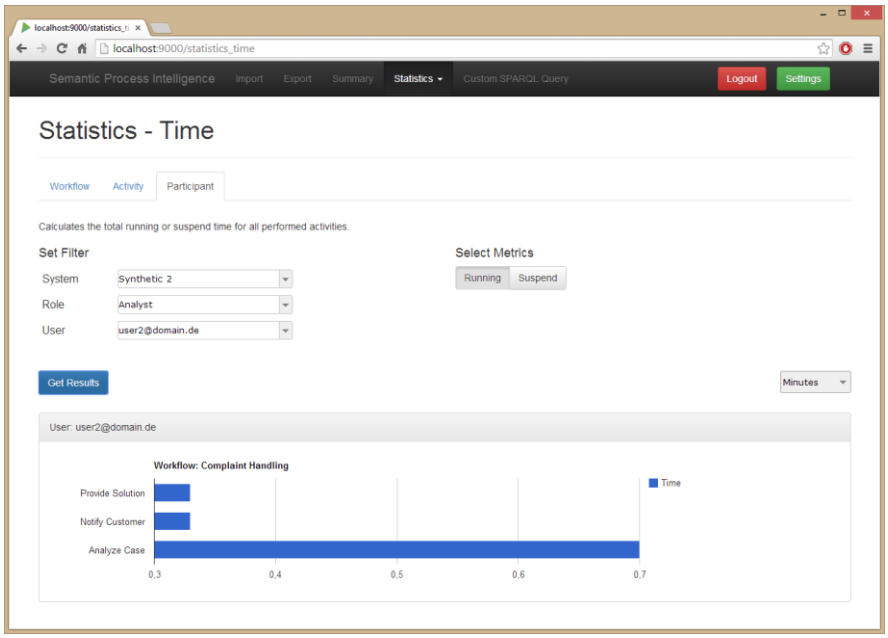


Figure 6-18 SPA – Statistics (Time/Participant)

The last statistic section (see Figure 6-19) provides information about the different execution paths traversed during the execution of a workflow. The result displays all available paths as a string of activities indicating their execution order as well as their frequency.

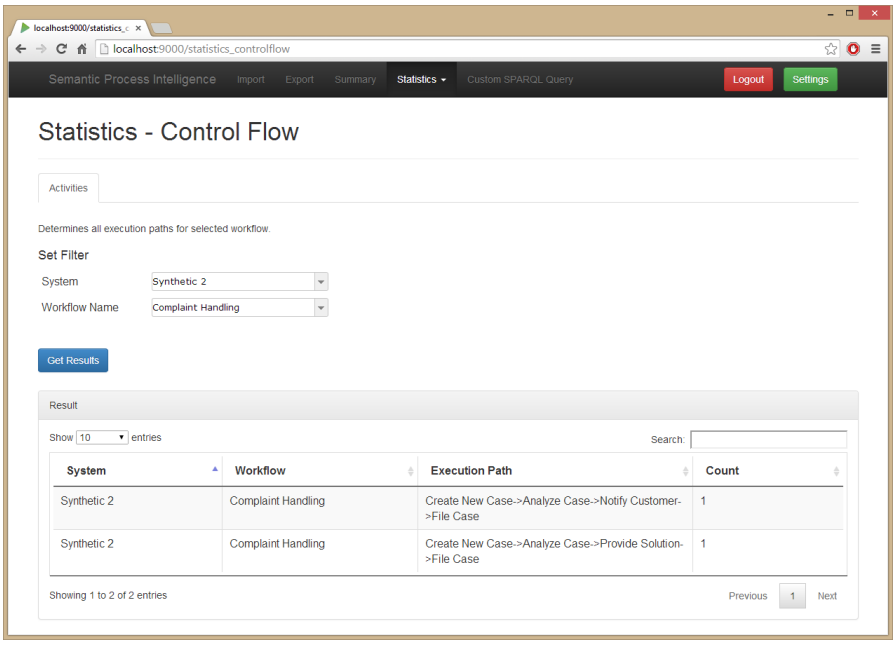


Figure 6-19 SPA – Statistics (Control Flow)

The area *Custom SPARQL Query* gives the user the ability to perform custom queries on the imported data and to extract further information that is not accessible through the provided statistic functions. The query editor includes syntax highlighting as well as syntax checking to accelerate the query development process. In addition, created queries can be saved for later usage or already saved queries can be opened. The queries are saved per user such that only the user that created a query can use or delete it (see Figure 6-20 and Figure 6-21).

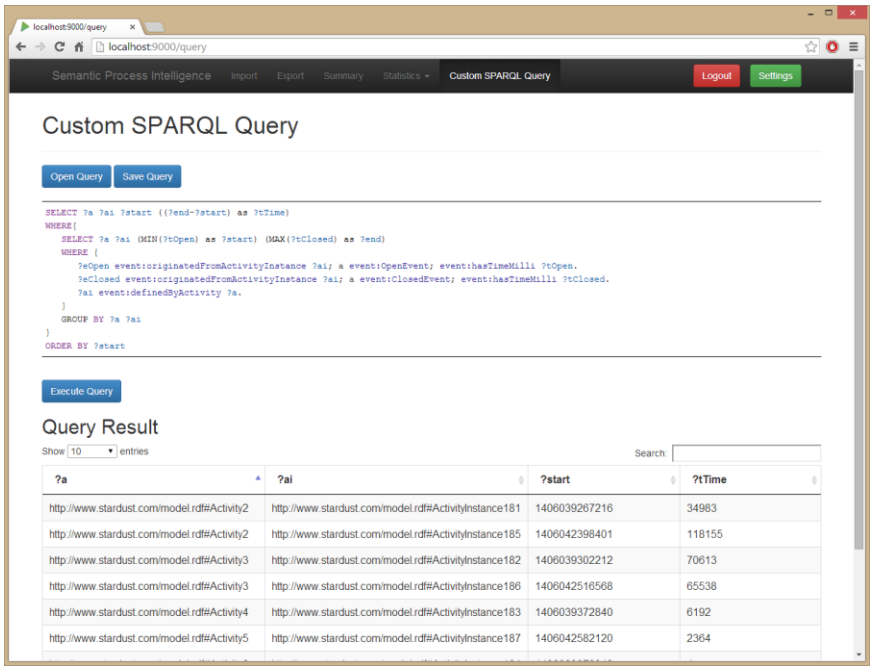


Figure 6-20 SPA – Custom SPARQL Query (Overview)

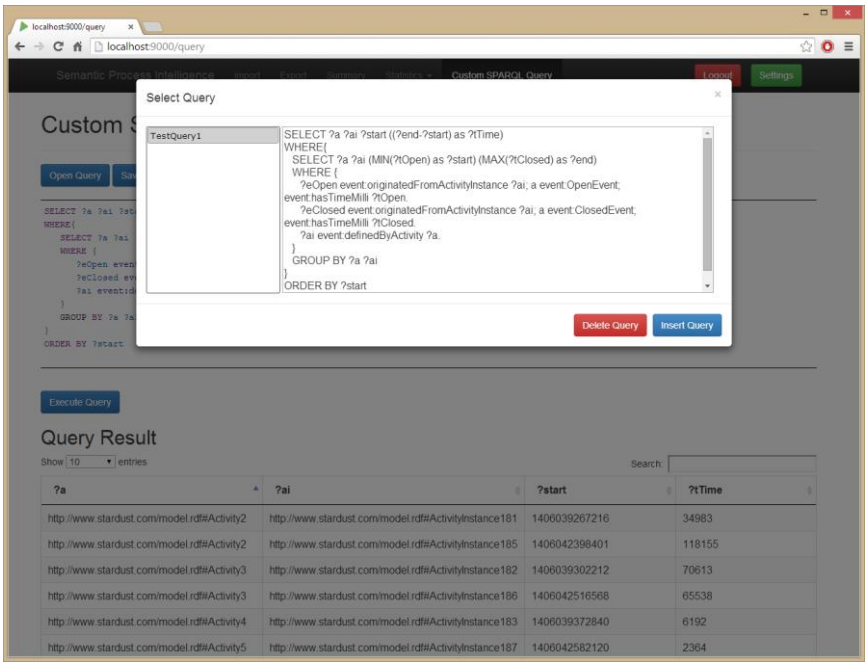


Figure 6-21 SPA – Custom SPARQL Query (Opening Queries)

The SPA also provides a settings section (see Figure 6-22) that lets a user easily change the ontology namespaces as well as URLs for download in case the initial ontologies are changed or moved to another location.

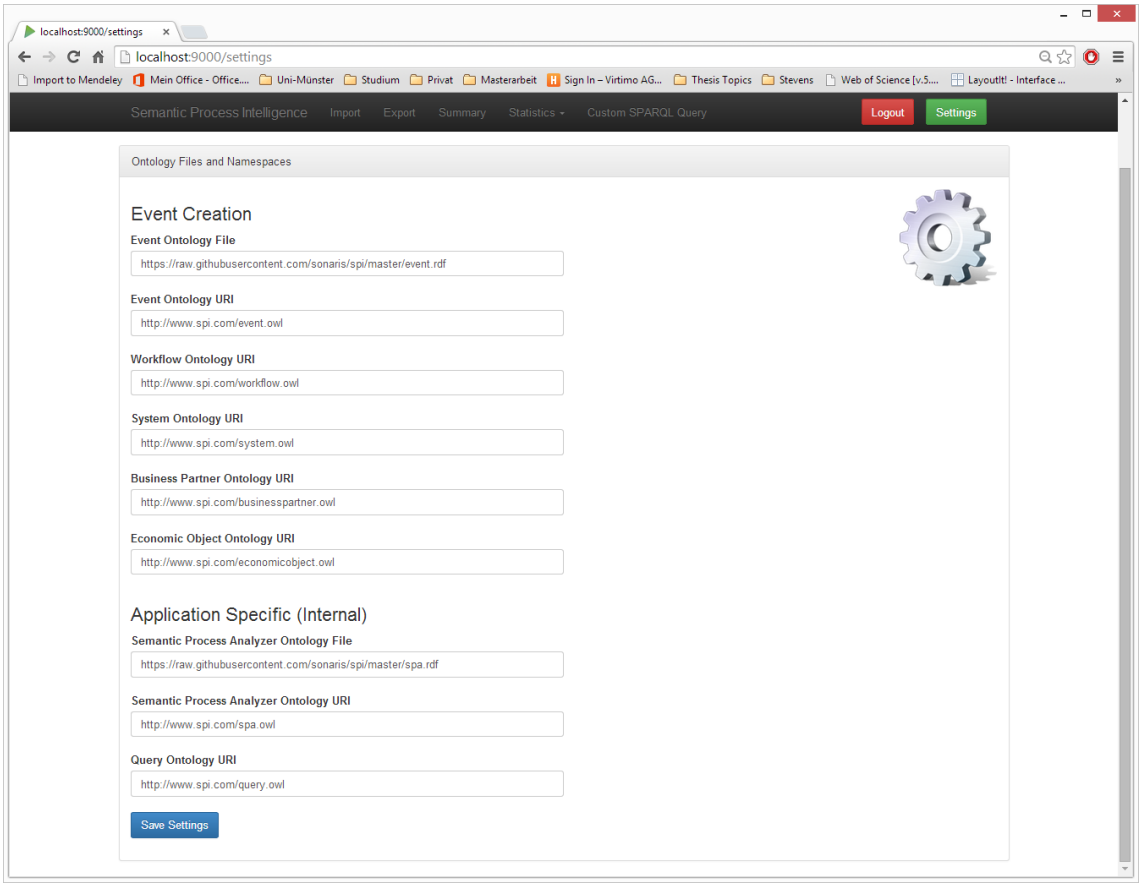


Figure 6-22 SPA – Settings

7 Prototype Demonstration and Evaluation

To demonstrate and evaluate the prototype, a set of specific test questions, based on the general competency questions and the workflows used for the synthetic event data, is derived in the beginning (section 7.1). In the following, the prototype is first demonstrated using the test questions and the synthetic event data (section 7.2) and then evaluated using the same set of questions but with event data from two real-life-BPMS (section 7.3). The prototype as well as the chosen BPMS were executed and accessed locally on a test system.

7.1 Test Questions

The following test questions were used for both the prototype demonstration and evaluation:

1. How many of the opened instances of the *New Order* and *Complaint Handling* workflow were completed?

2. What is the average, minimum and maximum turnaround time, running time and suspend time in minutes of activity *Analyze Case* in the *Complaint Handling* workflow?

3. How much time did the user accounts involved in the *Complaint Handling* workflow spend on their assigned tasks?

4. Which is the most commonly traversed execution path in the *Order* workflow?

5. Which customers ordered a product and later complained about it?

7.2 Synthetic Test

For the synthetic test, the workflow models and event data used for the ontology demonstration (see section 5.3) were used again. This time, however, the synthetic import function is used to directly import the created Excel files hiding the complexity of the created RDF data. In total, three Excel-files for the *Order* and three Excel files for the *Complaint Handling* workflow were created and had to be imported. Figure 7-1 shows the import settings for the *Order* workflow.

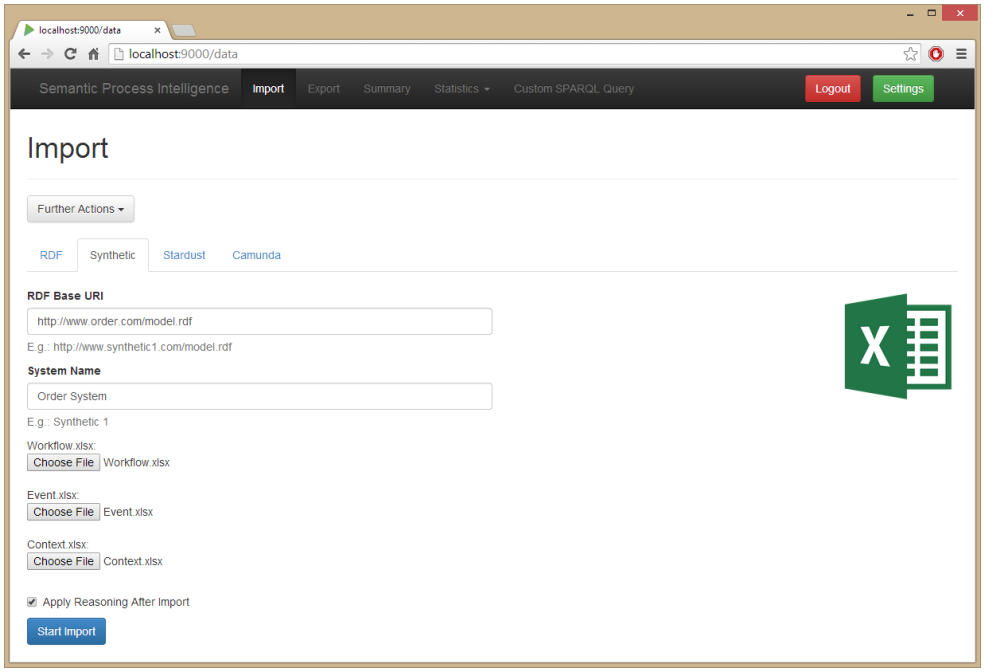


Figure 7-1 Synthetic Test – Import

After the import, the summary section (see Figure 7-2) is used to verify that all defined event data are imported as desired. The chart correctly shows that no Participants from the Order System were imported since its order workflow operates without human intervention.

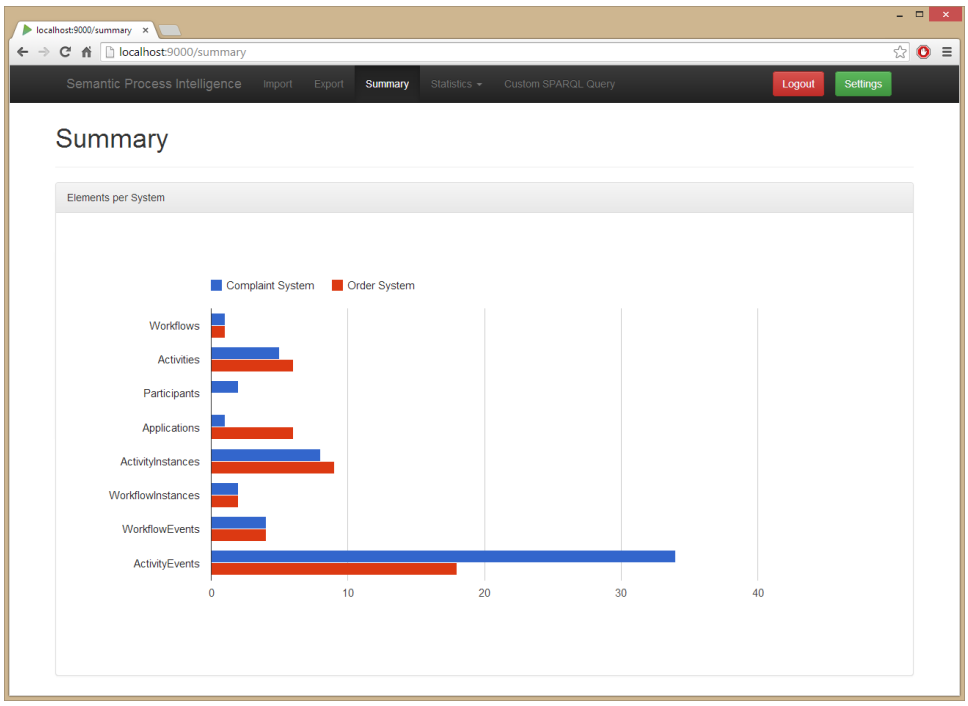


Figure 7-2 Synthetic Test – Summary

To answer test question (1), the quality section in the statistics area is opened. The generated charts correctly show that for the *New Order* as well as for the *Complaint Handling* workflow, all opened instances were also completed.

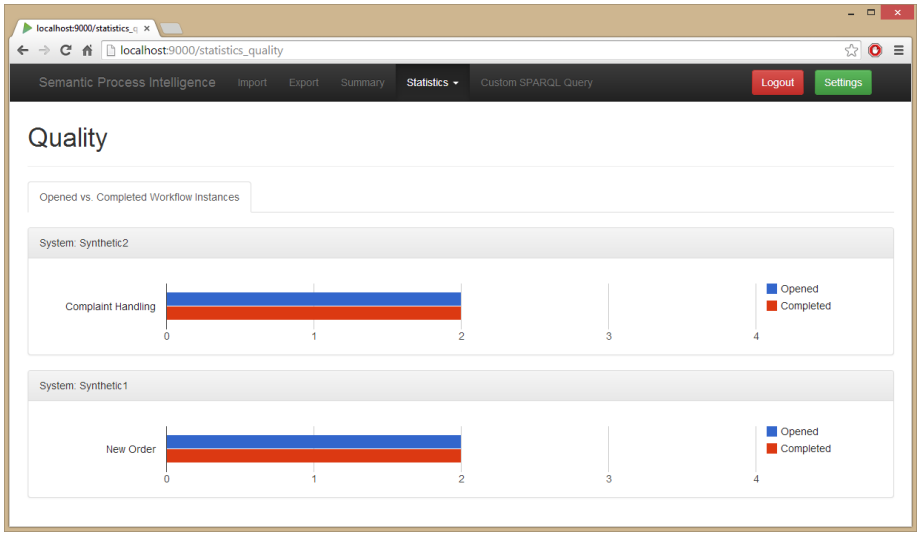


Figure 7-3 Synthetic Test – Answering Question 1

To answer test question (2), the activity section within the time area has to be opened. As filter criteria, the respective workflow and activity name have to be selected. Further, all available metrics have to be enabled and the time format has to be set to “Minutes”. Pressing the *Get Results* button returns the desired result which for instance reveals that on average (over all activity instances), the suspended time amounts to 0.2 minutes.

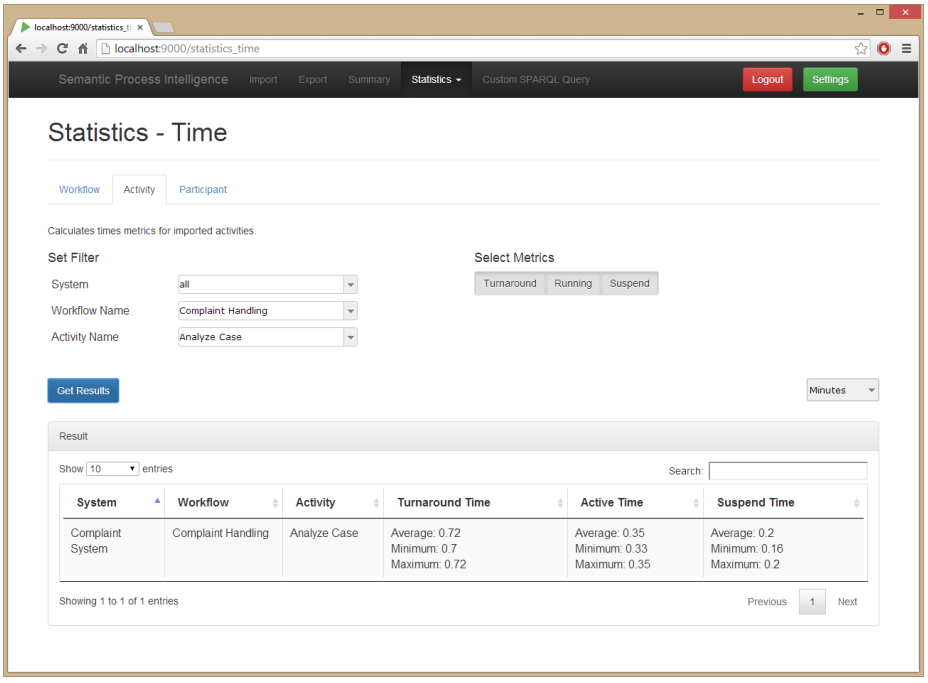


Figure 7-4 Synthetic Test – Answering Question 2

To answer the question which user has worked most on which activity (question 3), the activity section is opened and the system filter set to “Complaint System”. The results show that user *analyst1@domain.de* has worked most on activity *Analyze Case* (about 0.7 minutes) while *service_agent1@domain.de* was only responsible for working on activity *Create New Case*.

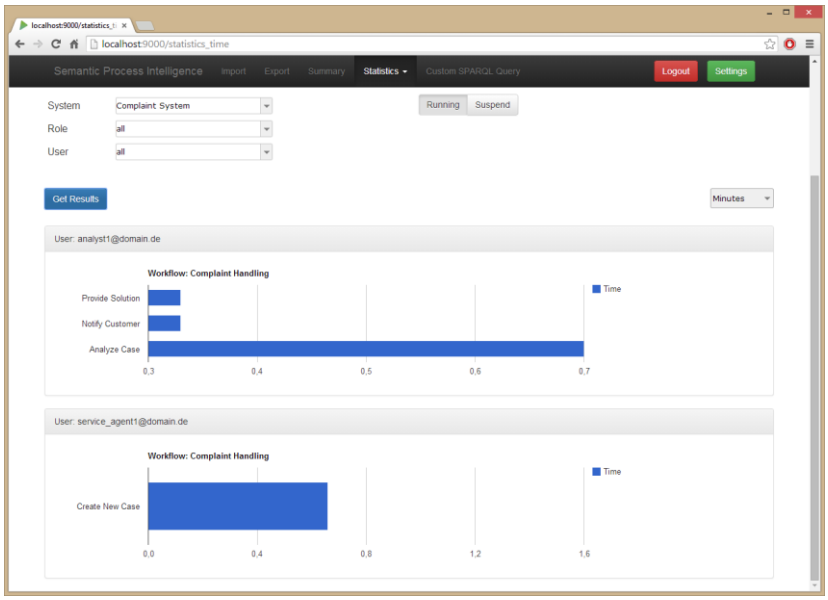


Figure 7-5 Synthetic Test – Answering Question 3

To answer the fourth question, the control flow area is opened and the filter set to system *Order System* and workflow *New Order*. The results show that two different execution paths have been traversed each time. The first path shows the case in which a new customer order could not be fulfilled while the second path shows the opposite case.

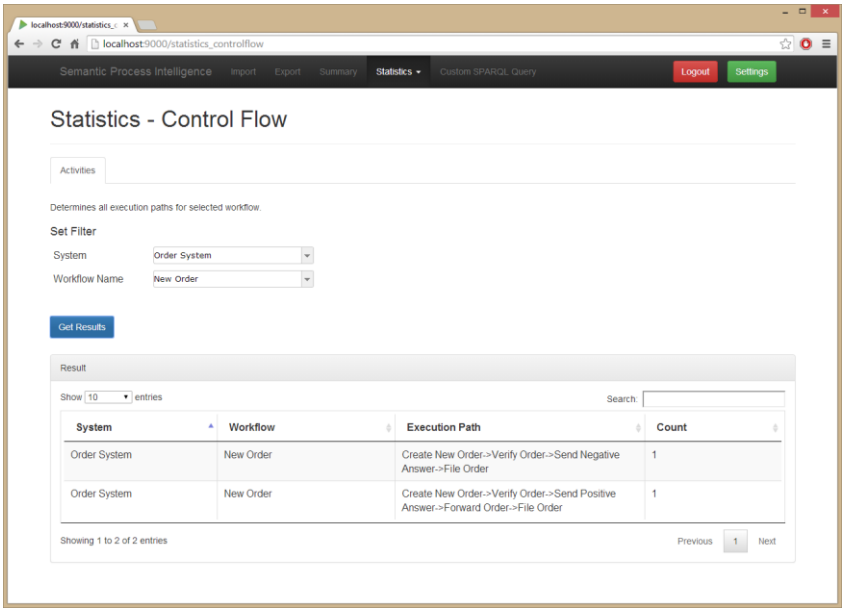


Figure 7-6 Synthetic Test – Answering Question 4

Test question (5) cannot be answered using the standard statistics functions since questions addressing two or more systems are not supported yet. Instead, a custom SPARQL query has to be created. In this case, Query 9, presented in section 5.3, can be reused. Executing the query returns the result depicted in Figure 7-7. For the real-life test in the following section, the query is saved under “cross-system question 1”.

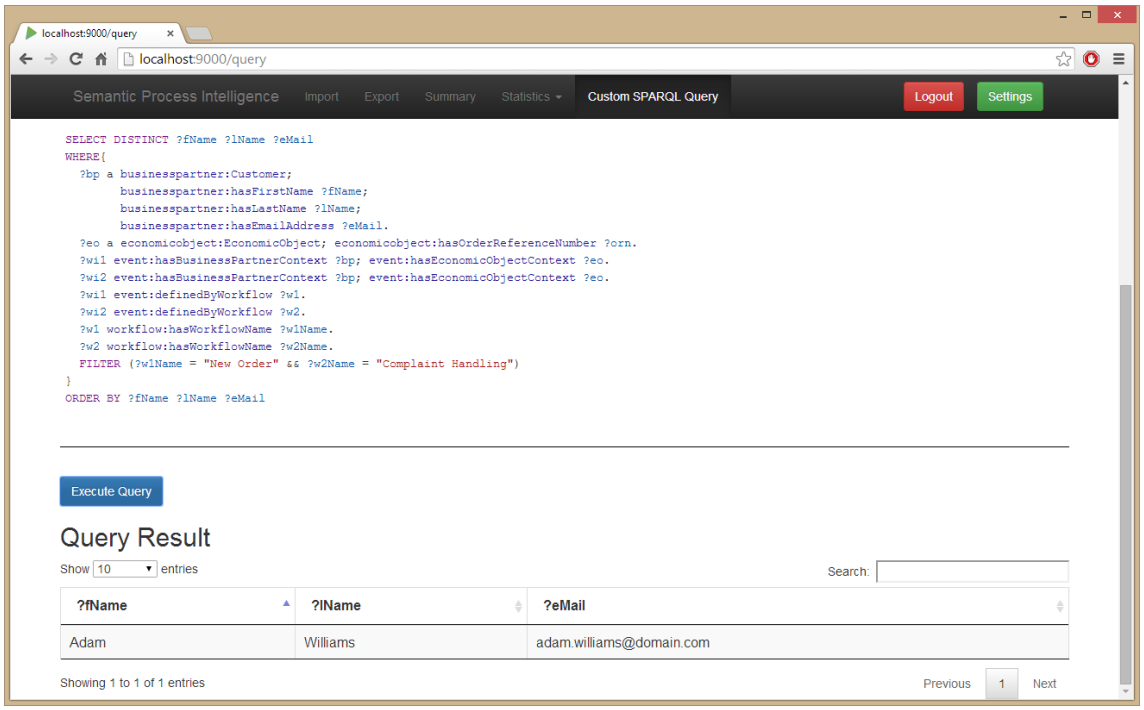


Figure 7-7 Synthetic Test: Answering Question 5

It can be seen that *Adam Williams* is the only customer that first ordered a product and then complained about it.

7.3 Real-Life Test

For the real-life test, the workflow models used for the synthetic test were transformed into executable workflows with the order workflow being executed on the Camunda BPMS and the complaint workflow on the Stardust BPMS. Before the analysis results will be shown, the workflow models and its specifics will be presented first.

7.3.1 Order Workflow (Camunda BPMS)

When presented in section 5.1, the order workflow was described as a completely automated workflow without human intervention. To simplify testing and generation of events, one manual step was introduced allowing for the creation of a new order via the Camunda web interface. The *Create New Order* start event was used for this purpose. To prescribe the data that can be entered at this step, a simple HTML form was created and

attached to the start event. In addition to prescribing the data that can be entered, the form also prescribes the variable names under which the entered values are saved into the audit trail data base. The naming convention introduced in section 6.3.2 was used as depicted in Figure 7-9. In Camunda, it is not possible to assign a specific user or role to a manual start event. Hence, it can be triggered by all user accounts. All automatic tasks were assigned to dummy Java applications that served the purpose to simulate execution time (1 second each). The application assigned to *Verify Order* additionally contained a function that randomly returned a positive or a negative result that is used by the XOR connector to decide which one of the following two paths should be followed. The negative result has a probability of 10%. Figure 7-8 depicts the workflow model as shown in the Camunda modeller.

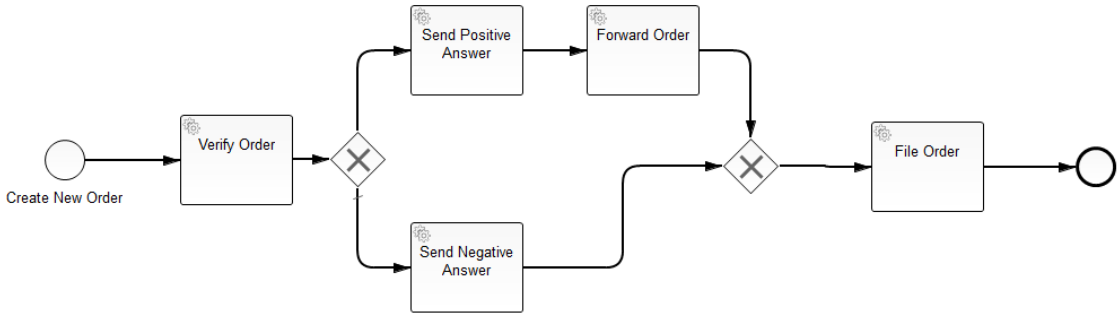


Figure 7-8 Real Life Test: Camunda Workflow Model

```
<form class="form-horizontal">
  <div class="control-group">
    <label class="control-label">Customer First Name</label>
    <div class="controls">
      <input form-field type="string" name="BusinessPartner#Person_Customer#FirstName"/>
    </div>
  </div>
  <div class="control-group">
    <label class="control-label">Customer Last Name</label>
    <div class="controls">
      <input form-field type="string" name="BusinessPartner#Person_Customer#LastName"/>
    </div>
  </div>
  <div class="control-group">
    <label class="control-label">Customer Email Address</label>
    <div class="controls">
      <input form-field type="string" name="BusinessPartner#Person_Customer#EmailAddress"/>
    </div>
  </div>
  <div class="control-group">
    <label class="control-label">Order Reference Number</label>
    <div class="controls">
      <input form-field type="string" name="EconomicObject#Order#OrderReferenceNumber"/>
    </div>
  </div>
</form>
```

Figure 7-9 Real Life Test: Camunda New Order Form

In total, the workflow was executed 10 times. For each workflow instance, the context data described in Table 7-1 was used. Those business partners that are marked in grey also complained later about their ordered product in the complaint workflow.

#	Name	Email	Order RN
1	Aaron Baker	aaron.baker@domain.com	1
2	Adam Williams	adam.williams@domain.com	2
3	Barry White	barry.white@domain.com	3
4	Bernard Finley	bernard.finley@domain.com	4
5	Chris Cornwell	chris.cornwell@domain.com	5
6	Randy Callahan	randy.callahan@domain.com	6
7	Greg Fisher	greg.fisher@domain.com	7
8	Mike Chavez	mike.chavez@domain.com	8
9	Scott Shepard	scott.shepard@domain.com	9
10	Will Finnegan	will.finnegan@domain.com	10

Table 7-1 Real Life Test: Camunda Context Data

7.3.2 Complaint Workflow (Stardust BPMS)

To make the complaint workflow executable in Stardust, appropriate data objects have to be defined according to the convention described in section 6.3.2. The objects, when connected appropriately, automatically generate simple forms that are displayed during the execution of the workflow in the Stardust web interface. The entered data are saved under the defined variable names depicted in Figure 7-11 and can later be extracted. Two additional fields *AnalysisResult* and *AnalysisSummary* were defined in the EconomicObject data object that were used for internal purposes. The first field is used to decide which of the two paths after the XOR connector should be followed while the second field serves as input for the steps *Provide Solution* and *Notify Customer*. A dummy application *FileCaseApp* is attached to activity *File Case*. Via the Stardust web interface, two user accounts with user names *service_agent1* and *analyst1* were defined and assigned to the *Service Agent* respectively the *Analyst* role. A manual trigger "New Case" was added to allow users to manually initialize a workflow instance. Figure 7-10 depicts the workflow model as shown in the Stardust modeler.

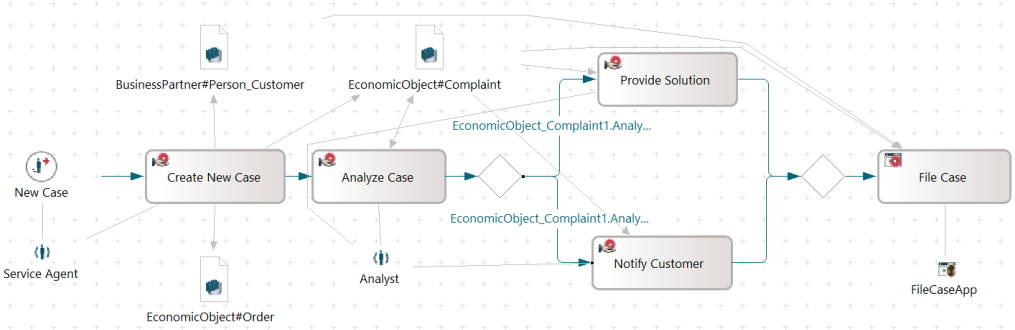


Figure 7-10 Real Life Test: Stardust Workflow Model

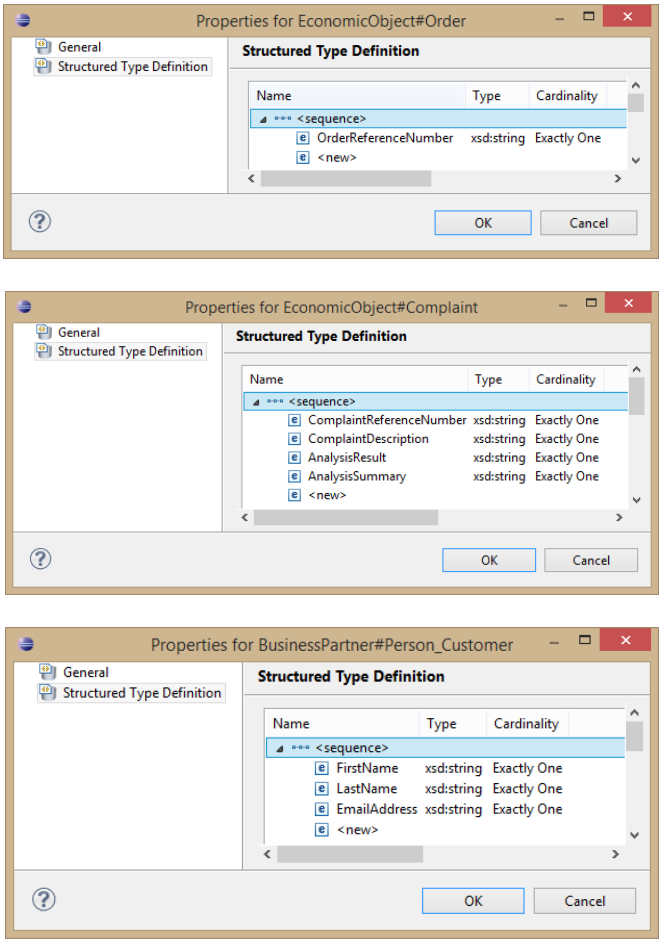


Figure 7-11 Real Life Test: Stardust Economic, Business Partner and Order Object

In total, the workflow was executed five times whereby the business partners used in the order workflow were reused for the complaint instances. Business partner *Adam Williams* was used twice to show two different complaints about the same product. Each time the *Analyze Case* activity was reached, it was suspended for a short time to be able to analyze suspend times with the SPA.

#	Name	Email	ComplaintRN	OrderRN
1	Adam Williams	adam.williams@domain.com	1	2
2	Chris Cornwell	chris.cornwell@domain.com	2	5
3	Randy Callahan	randy.callahan@domain.com	3	6
4	Adam Williams	adam.williams@domain.com	4	2
5	Will Finnegan	will.finnegan@domain.com	5	10

Table 7-2 Real Life Test: Stardust Context Data

7.3.3 Analysis

After both workflows were executed as described, the event data could be imported from both systems. Figure 7-12 and Figure 7-13 depict the import settings.

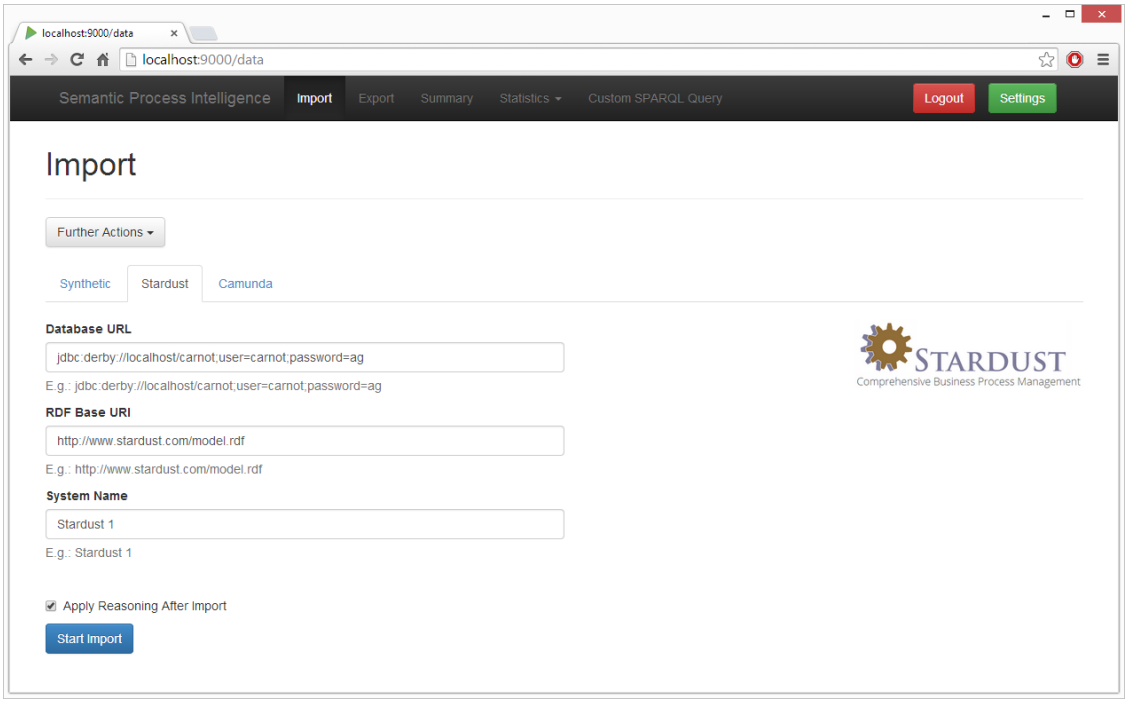


Figure 7-12 Real Life Test: Import Settings Stardust

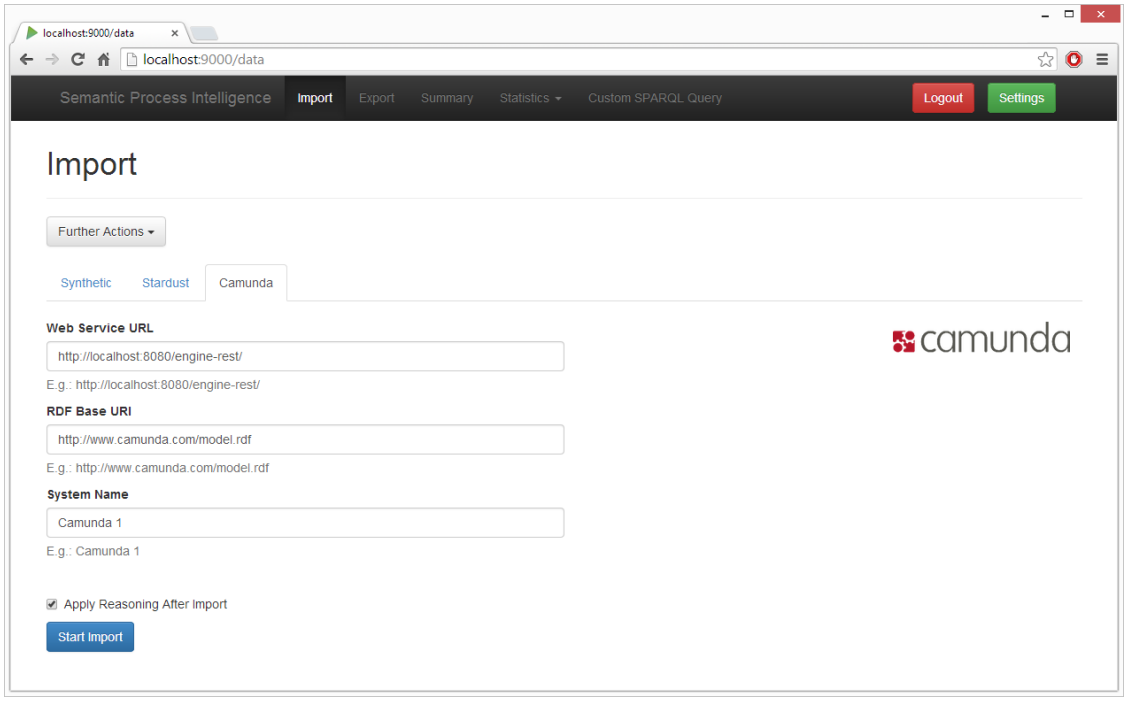


Figure 7-13 Real Life Test: Import Settings Camunda

To answer test question (1), the quality section in the statistics area is opened. The generated charts correctly show that for the *New Order* as well as for the *Complaint Handling* workflow, all opened instances were also completed. The other displayed

workflows stem from example workflows of the Camunda BPMN that were not completed yet.

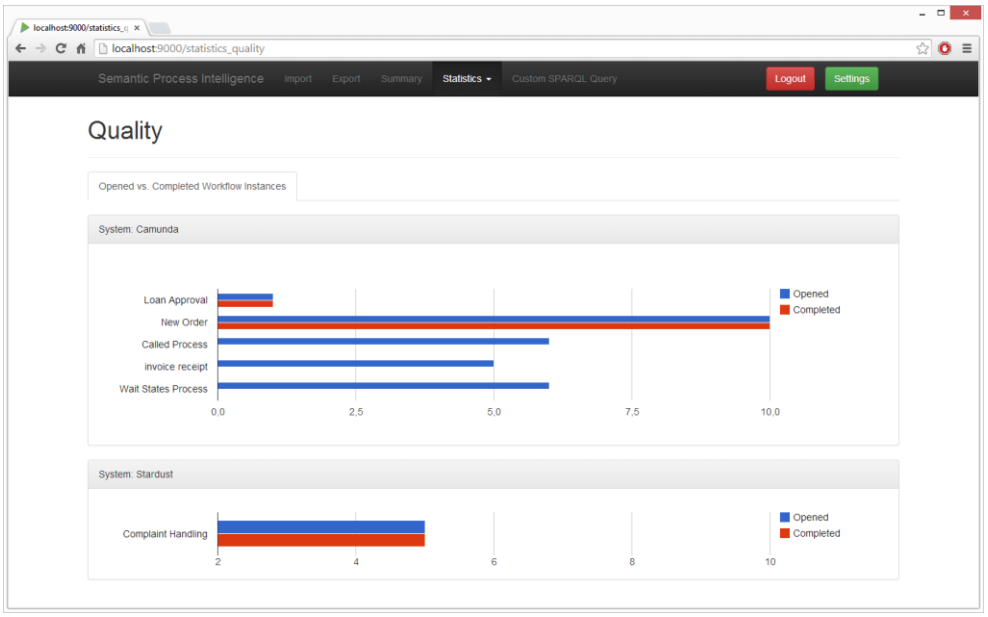


Figure 7-14 Real Life Test – Answering Question 1

To answer test question (2), the time area and its activity section have to be opened. As filter criteria, the respective workflow and activity name have to be selected. Further, all available metrics have to be selected and the time format has to be set to "Minutes". Pressing the *Get Results* button returns the desired result which, for instance, shows that on average (over all activity instances), 0.1 minutes are suspended and 0.53 minutes are running time.

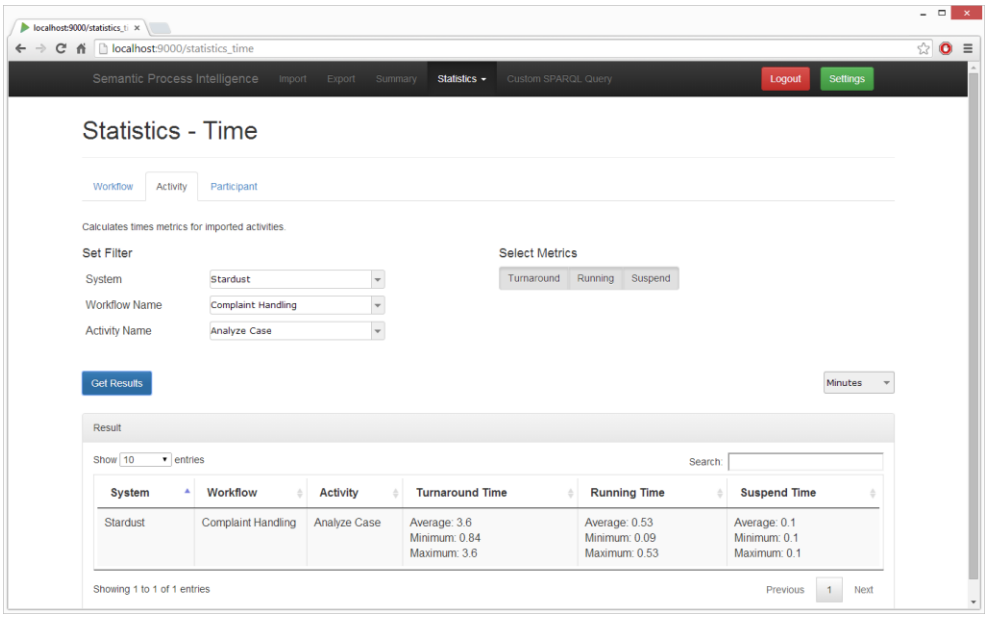


Figure 7-15 Real Life Test – Answering Question 2

To answer the question which user has worked most on which activity (question 3), the activity section is opened and the system filter set to “Complaint System”. The results show that user *analyst1* has worked most on activity *Analyze Case* (about 2.7 minutes) whereby *service_agent1* was only responsible for working on activity *Create New Case*.

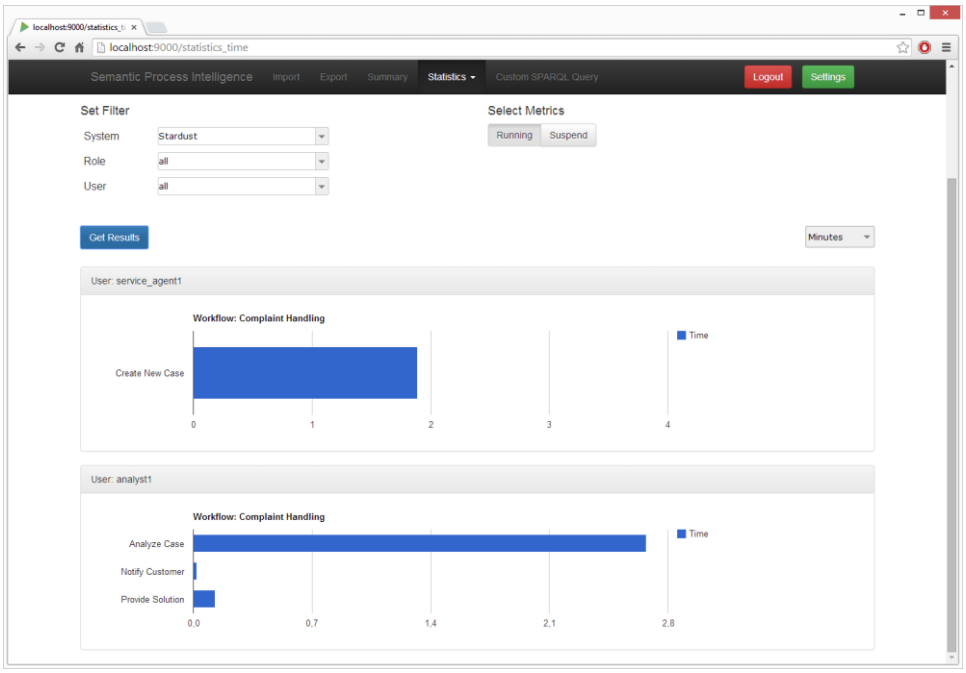


Figure 7-16 Real Life Test – Answering Question 3

To answer the fourth question, the control flow area is opened and the filter set to system *Order System* and workflow *New Order*. The results show that for all ten orders, the upper path of the workflow model was chosen and the order could be fulfilled.

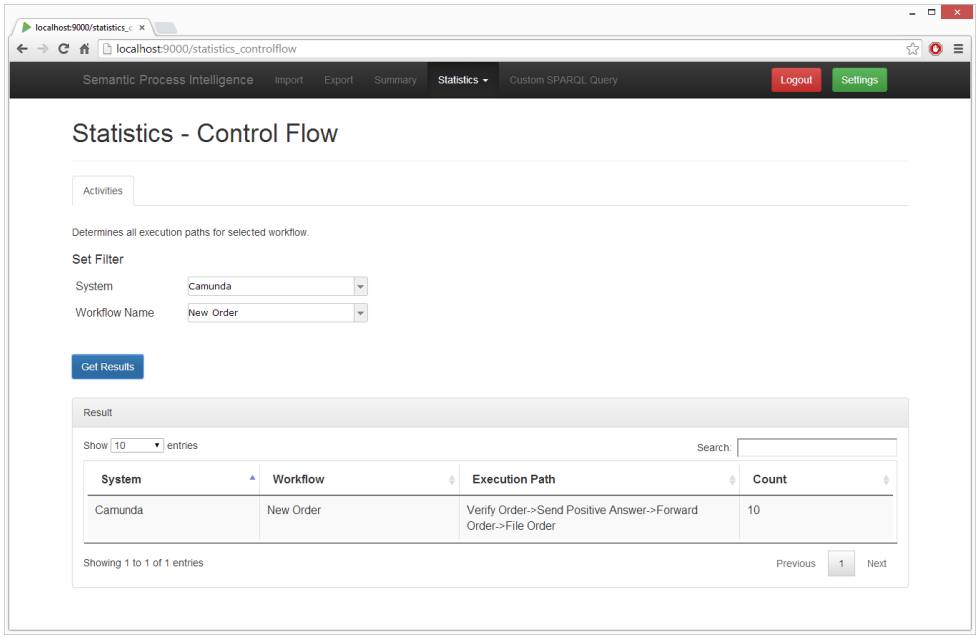


Figure 7-17 Real Life Test – Answering Question 4

For answering the test question 5, the custom SPARQL query created during the synthetic test is reused and inserted. Executing the query yields the following result which correctly contains those business partners highlighted in grey in Table 7-1.

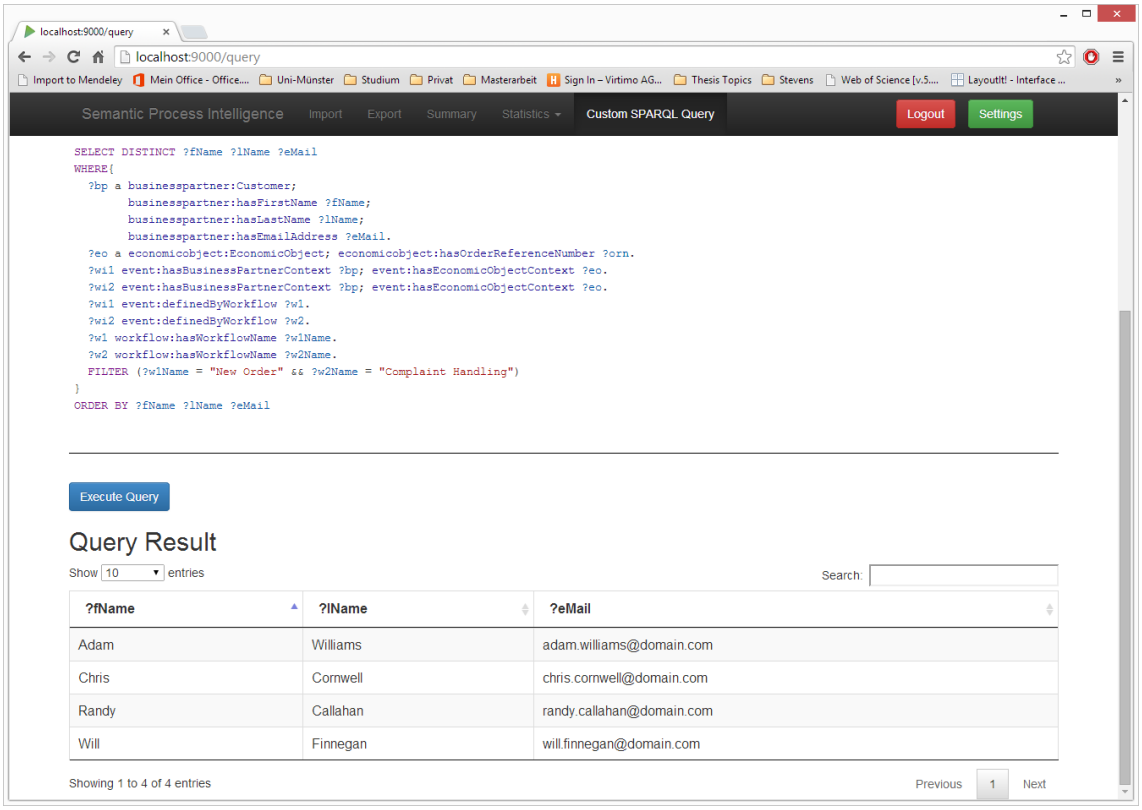


Figure 7-18 Real Life Test: Answering Question 5

8 Discussion and Outlook

Section 8.1 discusses the results of this thesis and highlights important and novel aspects of the presented approach. Section 8.2 summarizes important lessons learned both during the ontology and the prototype development while section 8.3 provides an outlook and implications for future research.

8.1 Discussion of Results

This thesis answers the raised research questions by developing a general business process event ontology that is used as part of a prototype for semantic process intelligence. The prototype shows how Semantic Web technologies can be used to collect, save and analyze arbitrary business process event data to answer the competency questions defined earlier.

The presented work started with investigating how business process events from arbitrary BPMS can be saved in an ontological representation. Therefore, existing event exchange and workflow/process exchange formats as well as existing ontologies were analyzed. Based on the analysis, four independent ontologies for describing workflows, systems, business partners and economic objects as well as an integrating event ontology were developed. Based on the event ontology, SPARQL queries were created that are able to answer typical questions in the area of business process intelligence.

Based on the ontology and the developed queries, a novel prototype for semantic process intelligence was developed. Part of the development process was the conceptualization of an application architecture that provides the necessary flexibility to import event data from arbitrary BPMS. The prototype itself is based on the popular semantic web framework *Apache Jena* as well as on the Java-based web development framework *Play*. It comprises functions for importing audit trail data from two different real-life BPMS (Stardust and Camunda) as well as from a proprietary format developed for this thesis. Furthermore, it contains analysis functions for quality-, time- and control-flow-related questions as well as a SPARQL query editor for developing and saving custom queries that can be used to answer questions that are not supported by the provided analysis functions. Additionally, imported event data can be exported into the common serialization formats used by the Semantic Web community.

Compared to traditional approaches for analyzing business process event data, the presented approach has four major advantages:

Convergence of exchange and analysis format: Since ontology-based event data can be efficiently queried using SPARQL, the need for an additional event exchange format like BPAF or MXML becomes obsolete. The same format can be used for saving and analyzing events, which accelerates the analysis processes and makes it less error-prone.

Avoidance of additional data integration: One of the main advantages of using ontologies is that instance data based on ontologies can directly be combined without the need for further data integration. This is due to the fact that the semantics are included with the data itself letting no room for interpretation. Would BPMS directly save their event data into the ontology-based format proposed in this thesis, these data could directly be combined and analyzed. At the moment, additional import components transferring the event data from the system-specific representation to the ontology-based one are still necessary.

Increased flexibility of the event model: Compared to data models used for relational databases, ontologies provide more flexibility to change or extent the existing model. In the Semantic Web world, anyone can say anything about any topic. New connections and new classes can be added without the necessity for redesigning the data model, changing foreign keys or adding null values. This facilitates the extension of the existing ontology and, for instance, adding new and more comprehensive context ontologies.

Simplified knowledge extraction: Using ontologies provides the advantage that automatic reasoning can be applied which can add missing information or infer additional connections between resources that haven't existed in the beginning. The prototype has shown how this principle can be used to ask cross-system questions that are based on combining event data from two or more systems in order to obtain additional valuable information. The example used in this thesis showed how customers can be identified that first ordered a product through one system and complaint about it through another system.

The presented prototype has two limitations. Limitation one is that although different workflow versions are supported by the ontology, the prototype only imports the newest one to simplify the analysis. This should be resolved in the future. Furthermore, the import of context data relies on the naming conventions presented in section 6.3.2 and fails if workflows are modeled differently in the respective tools. This, however, was necessary since the selected BPMS were not designed with semantic technologies in mind.

8.2 Lessons Learned

In the following, important lessons learned during the ontology and prototype development are presented.

Ontology Development

Understand the Semantic Web technology stack: The Semantic Web is complex and consists of a multitude of different technologies and concepts that are difficult to understand in the beginning. For researchers new to the field, it is important to understand the rationale behind the different Semantic Web technologies and how they are interlinked. This is especially important when Semantic Web frameworks such as Apache Jena are used that are built around the Semantic Web standards. Despite its complex and controversial nature, only few good information sources about the Semantic Web exist. Especially publicly available web sources are often not suited for an introduction since they are, in many cases, outdated, do not address beginners or are too simplifying. Two books that were of great help for the creation of this thesis were especially "A Developer's Guide to the Semantic Web" written by Liyang Yu and "Semantic Web for the Working Ontologist" written by Dean Allemang and James Hendler.

Every ontology serves a purpose: Although ontologies are supposed to capture "general truth", there are different ways to model an ontology for the same domain. When developing an ontology, the intended usage scenarios must be kept in mind and classes and properties must be designed in a way that the desired information can be extracted as easy as possible (through SPARQL queries).

Reasoning should only complement SPARQL: Although automatic reasoning can be powerful, it is only one way to extract information from a given RDF document. In fact, most information that can be extracted through reasoning can also be obtained by using SPARQL queries. A big disadvantage of reasoning is that it is slow. Already a relatively small number of triples like the ones created during the test cases in chapter 7, can lead to reasoning times of almost one minute. In application contexts where the data sources change regularly and responsiveness is important (which applies for process intelligence), reasoning should only be used when really necessary and SPARQL queries be favored.

Prototype Development

Avoid and hide complexity: The advantage of ontologies and RDF-documents is at the same time their disadvantage: unique URIs. Every resource that has to be created, needs a unique URI making it possible to unambiguously reference that resource. When creating RDF-documents, managing and creating URIs can be error prone and distract the developer from the actual goal of saving information. Hence, it is recommendable to develop an intermediate layer that hides as much as possible of the complexity.

Generate Example Data: Before starting with the prototype development, the created ontology should be tested with example queries to verify that it suits the purpose. To be

able to query it, example data is necessary. Protégé allows for the creation of example data and exporting them as an RDF-document. For small examples, this is sufficient. If, however, larger datasets with many different classes and properties have to be created, the manual creation in Protégé is time consuming and error-prone. In this case (which it was for this thesis), it is recommendable to first develop a small application that generates necessary example data. The developed generator is still integrated in the finished prototype as the "Synthetic Import".

Use a Java-based web development framework: When the goal is to create a web-based application using Apache Jena for its application logic, it is highly recommendable to use a java-based web development framework. It not only facilitates the integration of Jena (which is Java-based), it also reduces the overall complexity by making it possible to manage the entire application's code in one single project and simplifying the later deployment of the application.

Avoid using Protégé's SPARQL editor: An important part of developing a semantic web application is the creation of SPARQL queries that are used to extract the necessary information from the data source. Protégé provides a SPARQL editor that can be used to query example instance data created in the instance section. The current version, however, neither provides syntax highlighting, an undo/redo functionality nor does it include inferred statements in the query results. This makes developing appropriate queries very difficult. A better solution is for example the SPARQL endpoint Jena Fuseki which is part of the standard Jena framework. It provides all missing functionalities (except syntax highlighting) and an intuitive web interface for uploading RDF-documents and querying them with SPARQL.

Ontologies created with Protégé do not automatically work with Apache Jena: Although Jena supports OWL, it does not support documents saved in the native OWL format. Instance as well as ontology documents must be saved in the RDF/XML format in order to work with Jena. Another problem are import statements generated by Protégé: Protégé automatically assumes that an imported ontology's URI is dereferencable and represents the address where the ontology can be downloaded. If this is not the case, which it was for the prototype, the generated import statements in the RDF/XML export have to be changed manually. Otherwise, Jena cannot work with an ontology that references other ontologies.

8.3 Outlook and Future Research

The prototype has made several simplifications to accelerate development and to be able to show different aspects of semantic process intelligence. The first simplification was

that not the current version of XPDL (2.2) was used to create a workflow ontology but version 1.0 which is less complex but still sufficient for most use cases. A future task would be to create a fully XPDL 2.2 compliant workflow ontology. Further, only simplified context ontologies that served the purpose to illustrate the general principle of adding context to the workflow instance were used. Future research should focus on extending the provided ontologies and making them applicable for a broader range of use cases. At last, the developed user interface only provides a basic set of analysis functions that can be extended in the future.

One big disadvantage of working with RDF data compared to relational data is that it is more difficult for humans to work with it. This is on the one hand due to the fact that every kind of information is represented as a triple and on the other hand that good and free tools for dealing with large amounts of RDF-data are missing. In most cases, the only way to actually see and verify the created data is to create and examine an export into one of the various serialization formats. Some tools (including Protégé) allow for the creation of a visual graph model based on the data, which is, however, impractical for large data sets. Compared to dealing with structured tables in relational databases, this constitutes a major obstacle for using Semantic Web technology. This became also clear during the creation of the prototype. Therefore, it makes sense to put further research effort into the creation of better tools for dealing with semantic web data and making it accessible for a broader range of users.

Another problem of using semantic technologies is the query language SPARQL. While capable of answering most questions, it is in many cases less powerful than SQL or extended dialects like Transact-SQL. This makes many SPARQL queries verbose and more complex than comparable SQL queries. SPARQL is, however, much younger than SQL and will surely be extended in the future. The current version 1.1 published in 2013 already constitutes a major improvement to the previous version.

At the moment, no standardized BPMS exists that already saves business process event data in an ontology-based format making it necessary to convert its internal format to a new representation though separate importers. In case of Stardust, a separate investigation of the underlying audit trail database was necessary to be able to extract the right information. If process event data would be directly saved using the presented ontology, no further data transformation would be necessary and available data could be directly analyzed no matter from which source they were extracted. This would conform to the vision of the Semantic Web.

References

Van der Aalst, W. M. P. 1998. "The Application of Petri Nets to Workflow Management," *Journal of Circuits, Systems and Computers* (08:01), pp. 21–66.

Van der Aalst, W. M. P. 2012. "Process Mining: Overview and Opportunities," *ACM Transactions on Management Information Systems* (3:2), pp. 7:1–7:17.

Van der Aalst, W. M. P., de Beer, H. T., and van Dongen, B. F. 2005. "Process Mining and Verification of Properties: An Approach Based on Temporal Logic," in *Proceedings of the 2005 Confederated International Conference on the Move to Meaningful Internet Systems*, Lecture Notes in Computer Science, R. Meersman and Z. Tari (eds.), Springer Berlin Heidelberg, January 1, pp. 130–147.

Van der Aalst, W. M. P., ter Hofstede, A. H. M., and Weske, M. 2003. "Business Process Management: A Survey," in *Business Process Management*, Lecture Notes in Computer Science, W. M. P. van der Aalst and M. Weske (eds.), Springer Berlin Heidelberg, pp. 1–12.

Van der Aalst, W. M. P., and de Medeiros, A. 2005. "Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance," *Electronic Notes in Theoretical Computer Science* Proceedings of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models, (121), pp. 3–21.

Van der Aalst, W. M. P., Reijers, H. A., Weijters, A. J. M. M., van Dongen, B. F., Alves de Medeiros, A. K., Song, M., and Verbeek, H. M. W. 2007. "Business process mining: An industrial application," *Information Systems* (32:5), pp. 713–732.

Agarwal, P. 2005. "Ontological considerations in GIScience," *International Journal of Geographical Information Science* (19:5), pp. 501–536.

Allemang, D., and Hendler, J. 2011. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*, Elsevier.

Anderson, E., and Oliver, R. L. 1987. "Perspectives on behavior-based versus outcome-based salesforce control systems," *Journal of Marketing* (51:4), pp. 76–88.

Bächle, M., and Kirchberg, P. 2007. "Frameworks für das Web2.0," *Informatik-Spektrum* (30:2), pp. 79–83.

Baquero, A. V., and Molloy, O. 2013. "Integration of Event Data from Heterogeneous Systems to Support Business Process Analysis," in *Knowledge Discovery, Knowledge Engineering and Knowledge Management*, Communications in Computer and Information Science, A. Fred, J. L. G. Dietz, K. Liu, and J. Filipe (eds.), Springer Berlin Heidelberg, pp. 440–454.

Becker, J., and Kahn, D. 2003. "The Process in Focus," in *Process Management*, P. D. J. Becker, D. M. Kugeler, and P. D. M. Rosemann (eds.), Springer Berlin Heidelberg, pp. 1–12.

Becker, J., Matzner, M., Müller, O., and Walter, M. 2012. "A Review of Event Formats as Enablers of Event-Driven BPM," in *Business Process Management Workshops*, Lecture Notes in Business Information Processing, F. Daniel, K. Barkaoui, and S. Dustdar (eds.), Springer Berlin Heidelberg, pp. 433–445.

Bergamaschi, S., Beneventano, D., Guerra, F., and Orsini, M. 2011. "Data Integration," in *Handbook of Conceptual Modeling*, D. W. Embley and B. Thalheim (eds.), Springer Berlin Heidelberg, pp. 441–476.

Berners-Lee, T. 1998. "Relational Databases and the Semantic Web (in Design Issues)," available at http://www.w3.org/DesignIssues/RDB-RDF.html, retrieved April 23, 2014.

Berners-Lee, T., Hendler, J., and Lassila, O. 2001. "The Semantic Web," *Scientific American* , pp. 29–37.

Bizer, C., and Schultz, A. 2008. "Benchmarking the performance of storage systems that expose SPARQL endpoints," in *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems*.

Camunda. 2014. "Open Source BPM and Workflow with BPMN 2.0 | camunda BPM," available at http://camunda.org/, retrieved July 9, 2014.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K. 2004. "Jena: Implementing the Semantic Web Recommendations," in *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, New York, NY, USA, pp. 74–83.

Casati, F., Castellanos, M., Dayal, U., Hao, M. C., Sayal, M., and Shan, M.-C. 2002. "Business operation intelligence research at hp labs," *IEEE Data Engineering Bulletin* (25:4), pp. 32–35.

Castellanos, M., de Medeiros, A. K. A., Mendling, J., Weber, B., and Weijters, A. J. 2009. "Business Process Intelligence," in *Handbook of Research on Business Process Modeling*, Information Science Reference.

Chinosi, M., and Trombetta, A. 2012. "BPMN: An introduction to the standard," *Computer Standards & Interfaces* (34:1), pp. 124–134.

Corcho, O., Fernández-López, M., and Gómez-Pérez, A. 2003. "Methodologies, tools and languages for building ontologies. Where is their meeting point?," *Data & Knowledge Engineering* (46:1), pp. 41–64.

Corcho, O., Fernández-López, M., and Gómez-Pérez, A. 2006. "Ontological Engineering: Principles, Methods, Tools and Languages," in *Ontologies for Software Engineering and Software Technology*, C. Calero, F. Ruiz, and M. Piattini (eds.), Springer Berlin Heidelberg, pp. 1–48.

Corea, S., and Watters, A. 2007. "Challenges in Business Performance Measurement: The Case of a Corporate IT Function," in *Business Process Management*, Lecture Notes in Computer Science, G. Alonso, P. Dadam, and M. Rosemann (eds.), Springer Berlin Heidelberg, pp. 16–31.

Dumas, M., Aalst, W. M. van der, and Hofstede, A. H. ter. 2005. *Process-Aware Information Systems: Bridging People and Software Through Process Technology*, John Wiley & Sons.

EDMCouncil. 2014. "EDM Council Semantics Repository," available at http://www.edmcouncil.org/semanticsrepository/index.html, retrieved July 10, 2014.

Ferreira, D. R., Alves, S., and Thom, L. H. 2012. "Ontology-Based Discovery of Workflow Activity Patterns," in *Business Process Management Workshops*, Lecture Notes in Business Information Processing, F. Daniel, K. Barkaoui, and S. Dustdar (eds.), Springer Berlin Heidelberg, pp. 314–325.

Genrich, M., Kokkonen, A., Moormann, J., Muehlen, M. zur, Tregear, R., Mendling, J., and Weber, B. 2008. "Challenges for Business Process Intelligence: Discussions at the BPI Workshop 2007," in *Business Process Management Workshops*, Lecture Notes in Computer Science, A. ter Hofstede, B. Benatallah, and H.-Y. Paik (eds.), Springer Berlin Heidelberg, pp. 5–10.

Gontar, B., and Gontar, Z. 2012. "Event log standards in BPM domain," *Information Systems in Management* (1).

Gregor, S., and Hevner, A. R. 2013. "Positioning and presenting design science research for maximum impact," *MIS Quarterly* (37:2), pp. 337–356.

Gregory, S. A. 1966. "Design Science," in *The Design Method*, S. A. Gregory (ed.), Springer US, pp. 323–330.

Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., and Shan, M.-C. 2004. "Business Process Intelligence," *Computers in Industry* Process / Workflow Mining, (53:3), pp. 321–343.

Gruber, T. R. 1993. *A Translation Approach to Portable Ontology Specifications*, Stanford University Computer Science Department Knowledge Systems.

Haller, A., Gaaloul, W., and Marmolowski, M. 2008. "Towards an XPDL Compliant Process Ontology," in *Proceedings of the 2008 IEEE Congress on Services*, pp. 83–86.

Havey, M. 2005. *Essential Business Process Modeling*, O'Reilly Media.

Heravi, B. R., Bell, D., Lycett, M., and Green, S. D. 2010. "Towards an Ontology for Automating Collaborative Business Processes," in *Proceedings of the 14th Enterprise Distributed Object Computing Conference*, pp. 311–319.

Horridge, M., Parsia, B., and Sattler, U. 2009. "Explaining Inconsistencies in OWL Ontologies," in *Scalable Uncertainty Management*, Springer, pp. 124–137.

Janiesch, C., Matzner, M., and Müller, O. 2011. "A Blueprint for Event-Driven Business Activity Management," in *Business Process Management*, Lecture Notes in Computer Science, S. Rinderle-Ma, F. Toumani, and K. Wolf (eds.), Springer Berlin Heidelberg, pp. 17–28.

Jarrar, M. 2008. "Towards Effectiveness and Transparency in e-Business Transactions - An Ontology for Customer Complaint Management," in *Semantic Web Methodologies for E-Business Applications*, Idea Group.

Jena. 2014. "Apache Jena - Getting started with Apache Jena," (available at https://jena.apache.org/getting_started/index.html; retrieved July 16, 2014).

Lawrence, P., Bouzeghoub, M., Fabret, F., and Matulovic-broqué, M. 1997. *Workflow Handbook*, Wiley.

Martinez-Cruz, C., Blanco, I. J., and Vila, M. A. 2012. "Ontologies Versus Relational Databases: Are They So Different? A Comparison," *Artificial Intelligence Review* (38:4), pp. 271–290.

McLellan, M. 1996. "Workflow Metrics - One of the great benefits of Workflows," in *Praxis des Workflow-Managements*, H. Österle and P. Vogler (eds.), Vieweg+Teubner Verlag, pp. 301–318.

Meditskos, G., and Bassiliades, N. 2010. "DLEJena: A practical forward-chaining OWL 2 RL reasoner combining Jena and Pellet," *Web Semantics: Science, Services and Agents on the World Wide Web* (8:1), pp. 89–94.

Moran, C. 2013. "Business-Value-of-Semantic-Technology.pdf," *semanticweb.com*, available at http://static59.mediabistro.com/content/Business-Value-of-Semantic-Technology.pdf, retrieved April 23, 2014.

Zur Muehlen, M., and Hansmann, H. 2012. "Workflowmanagement," in *Prozessmanagement*, J. Becker, M. Kugeler, and M. Rosemann (eds.), Springer Berlin Heidelberg, pp. 367–400.

Zur Muehlen, M., and Shapiro, R. 2010. "Business Process Analytics," in *Handbook on Business Process Management 2*, International Handbooks on Information Systems, J. vom Brocke and M. Rosemann (eds.), Springer Berlin Heidelberg, pp. 137–157.

Zur Muehlen, M., and Swenson, K. D. 2011. "BPAF: A Standard for the Interchange of Process Analytics Data," in *Business Process Management Workshops*, Springer, pp. 170–181.

Nicola, A. D., Lezoche, M., and Missikoff, M. 2007. "An Ontological Approach to Business Process Modeling," in *Proceedings of the 3rd Indian International Conference on Artificial Intelligence, pp 1794-1813*.

Norin, R. 2002. "Workflow Process Definition Interface: XML Process Definition Language.," Workflow Management Coalition (available at http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf).

Otto, A. 2003. "Supply Chain Event Management: Three Perspectives," *International Journal of Logistics Management* (14:2), pp. 1–13.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. 2007. "A design science research methodology for information systems research," *Journal of management information systems* (24:3), pp. 45–77.

Play. 2014a. "Usability," available at http://www.playframework.com/documentation/1.0.3.2/usability, retrieved July 17, 2014.

Play. 2014b. "Play Framework - Build Modern & Scalable Web Apps with Java and Scala," available at http://www.playframework.com/, retrieved July 17, 2014.

Play. 2014c. "IDE," available at http://www.playframework.com/documentation/2.3.x/IDE, retrieved July 17, 2014.

Protege. 2014. "Protégé," available at http://protege.stanford.edu/products.php, retrieved July 7, 2014.

ProtegeWiki. 2014. "WebProtege Release Notes," available at http://protegewiki.stanford.edu/wiki/WebProtegeReleaseNotes, retrieved July 7, 2014.

Quilitz, B., and Leser, U. 2008. "Querying Distributed RDF Data Sources with SPARQL," in *The Semantic Web: Research and Applications*, Lecture Notes in Computer Science, S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis (eds.), Springer Berlin Heidelberg, pp. 524–538.

Rinne, M. 2012. "SPARQL Update for Complex Event Processing," in Proceedings of the 11th International Semantic Web Conference, Lecture Notes in Computer Science, P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist (eds.), Springer Berlin Heidelberg, pp. 453–456.

Rinne, M., Abdullah, H., Törmä, S., and Nuutila, E. 2012. "Processing Heterogeneous RDF Events with Standing SPARQL Update Rules," in *On the Move to Meaningful Internet Systems: OTM 2012*, Springer, pp. 797–806.

Rozinat, A., and van der Aalst, W. M. P. 2008. "Conformance checking of processes based on monitoring real behavior," *Information Systems* (33:1), pp. 64–95.

Rücker, B. 2008. "Process Execution Languages," (available at http://camunda.com/download/JavaSpektrum0408_ruecker_process_execution_languages.pdf; retrieved May 26, 2014).

SAP. 2014. "SAP Business Partner - SAP Library," available at http://help.sap.com/saphelp_SNC700_ehp02/helpdata/en/b7/660736f95a6103e10000009b38f839/frameset.htm, retrieved July 10, 2014.

Schaaf, M., Grivas, S. G., Ackermann, D., Diekmann, A., Koschel, A., and Astrova, I. 2012. "Semantic complex event processing," *Recent Researches in Applied Information Science*, pp. 38–43.

Signavio. 2013. “Open Source BPM: camunda forks Activiti,” available at http://www.signavio.com/news/open-source-bpm-camunda-forks-activiti-2/, retrieved July 19, 2014.

Simon, H. A. 1996. *The Sciences of the Artificial*, MIT Press.

Soldic-Aleksic, J., and Stankic, R. 2011. “Business Intelligence,” in *International Encyclopedia of Statistical Science*, M. Lovric (ed.), Springer Berlin Heidelberg, pp. 188–189.

Stardust. 2013. “Stardust - A Comprehensive Business Process Management Suite,” available at http://www.eclipse.org/community/eclipse_newsletter/2013/june/article3.php, retrieved July 19, 2014.

Stardust. 2014a. “Stardust - History,” available at http://www.eclipse.org/stardust/general/history.php, retrieved July 19, 2014.

Stardust. 2014b. “Audit Trail Model,” available at http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.stardust.docs.dev%2Fhtml%2Fhandbooks%2Foperation%2Faudit-trail%2Fag-appendix-audittrail-1.htm&cp=58_7_1_5_0, retrieved July 19, 2014.

Stardust. 2014c. “Workflow Instance States,” available at http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.stardust.docs.dev%2Fhtml%2Fconcepts%2Fworkflow-basics%2Fmg-basics-4.htm, retrieved July 19, 2014.

Stardust. 2014d. “Activity Instance States,” available at http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.stardust.docs.dev%2Fhtml%2Fconcepts%2Fworkflow-basics%2Fmg-basics-5.htm&cp=58_4_0_4, retrieved July 19, 2014.

Sunkle, S., Kulkarni, V., and Roychoudhury, S. 2013. “Analyzable Enterprise Models Using Ontology,” in *CAiSE Forum*, (Vol. 998), pp. 33–40.

Uschold, M., and Gruninger, M. 1996. “Ontologies - Principles, Methods and Applications,” The Knowledge Engineering Review (11:2), pp.93-136.

Vysniauskas, E., Nemuraite, L., and Sukys, A. 2010. “A Hybrid Approach for Relating OWL 2 Ontologies and Relational Databases,” in *Perspectives in Business Informatics Research*, Lecture Notes in Business Information Processing, P. Forbrig and H. Günther (eds.), Springer Berlin Heidelberg, pp. 86–101.

W3C. 1999. “Resource Description Framework (RDF) Model and Syntax Specification,” available at http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, retrieved June 30, 2014.

W3C. 2006. “Linked Data - Design Issues,” available at http://www.w3.org/DesignIssues/LinkedData.html, retrieved July 3, 2014.

W3C. 2012. “OWL 2 Web Ontology Language Document Overview (Second Edition),” available at http://www.w3.org/TR/owl2-overview/, retrieved July 7, 2014.

W3C. 2014a. “W3C Semantic Web Activity,” available at http://www.w3.org/2001/sw/, retrieved July 2, 2014.

W3C. 2014b. “RDF 1.1 Primer,” available at http://www.w3.org/TR/rdf11-primer/, retrieved June 27, 2014.

W3C. 2014c. “RDF Current Status,” available at http://www.w3.org/standards/techs/rdf#w3c_all, retrieved August 7, 2014.

Webster, J., and Watson, R. T. 2002. “Analyzing the Past to Prepare for the Future: Writing a Literature Review,” *MIS Quarterly* (26:2), pp. 13–23.

WFMC. 2014a. “What is BPM? - Workflow Management Coalition,” available at http://www.wfmc.org/what-is-bpm, retrieved June 25, 2014.

WFMC. 2014b. “XPDL,” available at http://www.wfmc.org/standards/xpdl, retrieved May 26, 2014.

Wikipedia. 2014a. “Unix time - Wikipedia, the free encyclopedia,” available at http://en.wikipedia.org/wiki/Unix_time, retrieved July 10, 2014.

Wikipedia. 2014b. “Camunda BPM – Wikipedia,” available at http://de.wikipedia.org/wiki/Camunda_BPM#cite_note-4, retrieved July 19, 2014.

xpdl.org. 2014. “XPDL,” available at http://www.xpdl.org/, retrieved May 26, 2014.

XPDL.ORG. 2014. “XPDL 2.2,” available at http://www.xpdl.org/standards/xpdl-2.2/XPDL%202.2%20(2012-08-30).pdf, retrieved July 9, 2014.

Yu, L. 2011. *A Developer’s Guide to the Semantic Web*, Springer Berlin Heidelberg.

Zhu, T., Bakshi, A., Prasanna, V. K., and Gomadam, K. 2010. “Applying Semantic Web Techniques to Reservoir Engineering: Challenges and Experiences from Event Modeling,” in *Proceedings of the 7th International Conference on Information Technology: New Generations (ITNG)*, pp. 586–591.

Appendix

A Contents of the Digital Appendix

The following files and folders were uploaded into a separate GitHub repository (https://github.com/sonaris/spa) in order to enable a verification of the presented work as well as to foster future research in this domain.

- Thesis as *.pdf file
- *SPA IntelliJ Project*
- *Ontologies*
    - workflow.rdf
    - system.rdf
    - businesspartner.rdf
    - economicobject.rdf
    - event.rdf
    - query.rdf
    - spa.rdf
    - Overview.vsdx (overview of the combined event ontology)
- *Synthetic Test*
    - *Complaint Workflow*: Excel files describing the *New Order* workflow
    - *Order Workflow*: Excel files describing the *Complaint* workflow
    - *SPA Exports*: Exports from prototype
- *Real-Life Test*
    - *Stardust*
        - *Workspace*: Eclipse workspace containing the projects files and the Apache Server for deployment
    - *Camunda*
        - *Server*: Apache Server for deployment
        - *Workspace*: Eclipse workspace containing the project files
    - *SPA Exports*: Export from prototype

B Initial Literature Review

Using the search query described in section 4.3.1, Google Scholar returned 920 and SpringerLink 92 results. During a subsequent screening process of abstract and title 16 respectively 5 sources could be identified as relevant. Merging the results yielded 17 sources due to duplicates. An additional evaluation of the content further reduced the number of relevant sources to 8 which could be increased to 10 through an additional forward and backward search.

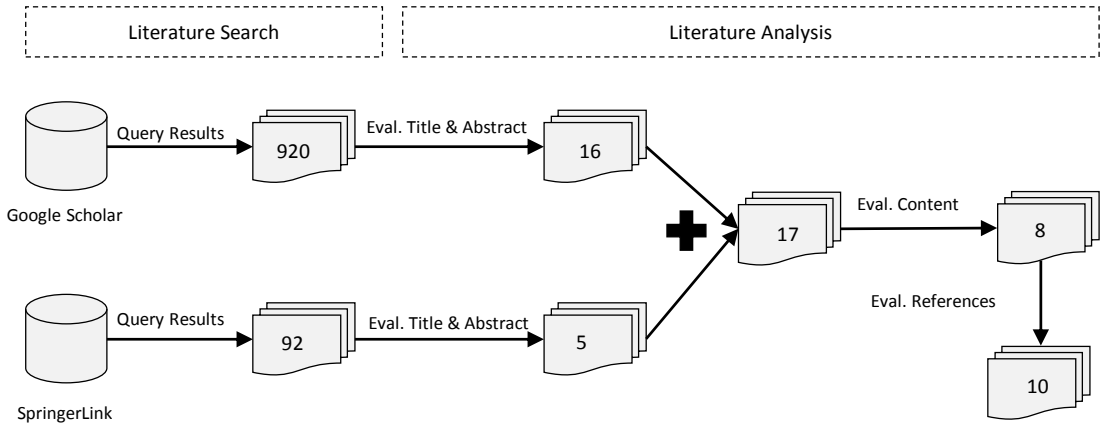


Figure B-1 Initial Literature Review Process.

The following table lists and summarizes the sources identified after the final evaluation step

#	Source	Short Summary
1	Becker et al. 2012	Systematic review of existing event formats and derivation of general requirements for edBPM event format.
2	zur Muehlen and Swenson 2011	Development of a Business Process Analytics Format (BPAF) for exchanging and analyzing business process events. Development of a general event state model.
3	Haller et al. 2008	Development of an XPDL compliant process ontology for process exchange and analysis.
4	Zhu et al. 2010	Use of semantic technologies to model and analyze event data in the reservoir engineering domain.

5	Schaaf et al. 2012	Analysis of application potentials for semantic technologies for complex event processing.
6	Rinne 2012	Potential of SPARQL and RDF for complex event processing.
7	Rinne et al. 2012	Potential of SPARQL and RDF for complex event processing.
8	Gontar and Gontar 2012	Review of existing event formats in the BPM domain.
9	zur Muehlen and Shapiro 2010	Overview of Business Process Analytics/Business Process Intelligence and common application areas.
10	Baquero and Molloy 2013	A prototype for event data integration from different heterogeneous system (*not* based on semantic technologies).

Table B-1 Short Summary of Identified Sources (Initial Literature Review)

C Event Independent Queries

The following additional queries were executed on the synthetic event used for the synthetic test.

Which activities precede the activity “Send Positive Answer” in the Workflow “New Order”?

```
SELECT ?predecessor_name
WHERE {
   ?a workflow:precededByActivity ?predecessor.
   ?a workflow:hasActivityName "Send Positive Answer".
   ?a workflow:partOfWorkflow ?w.
   ?w workflow:hasWorkflowName "New Order".
   ?predecessor workflow:hasActivityName ?predecessor_name.
}
```

?predecessor_name
Create New Order
Verify Order

Query 10 Event Independent Query 1

Is an activity “Notify Customers” preceded by an activity “Analyze Case” in the Complaint Handling Workflow?

```
ASK
WHERE {
   ?a1 workflow:precededByActivity ?a2.
   ?a1 workflow:hasActivityName "Notify Customer".
   ?a2 workflow:hasActivityName "Analyze Case".
   ?a1 workflow:partOfWorkflow ?w.
   ?w workflow:hasWorkflowName "Complaint Handling".
}
```

true

Query 11 Event Independent Query 2

How many different roles are involved in the Workflow “Complaint Handling”?

```
SELECT DISTINCT ?pName
WHERE {
   ?w workflow:hasWorkflowName "Complaint Handling".
   ?a workflow:partOfWorkflow ?w.
   ?a workflow:hasParticipantPerformer ?p.
   ?p workflow:hasParticipantName ?pName.
}
```

?pName
Analyst
ServiceAgent

Query 12 Event Independent Query 3

D Code Statistics

In total 10 java classes, 11 HTML pages and 12 JavaScript files were additionally created to the files provided by the Play- and other used frameworks. The following table summarizes the newly created lines of code including line breaks for better readability:

Type	Lines of Code
Java Classes	2009
HTML Files	1083
JavaScript Files	2542
CSS Files	47

Table D-2 Created Lines of Code

E Excel Files (Synthetic Import)

The following screenshots depict the Excel files used for the synthetic import of the complaint workflow.

The following 8 figures the content of Workflow.xlsx.

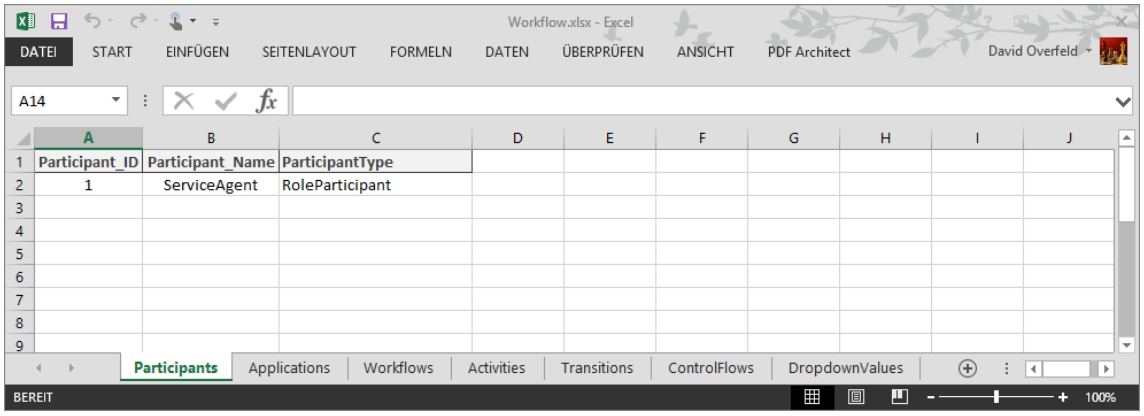


Figure E-2 Workflow.xlsx – Participants

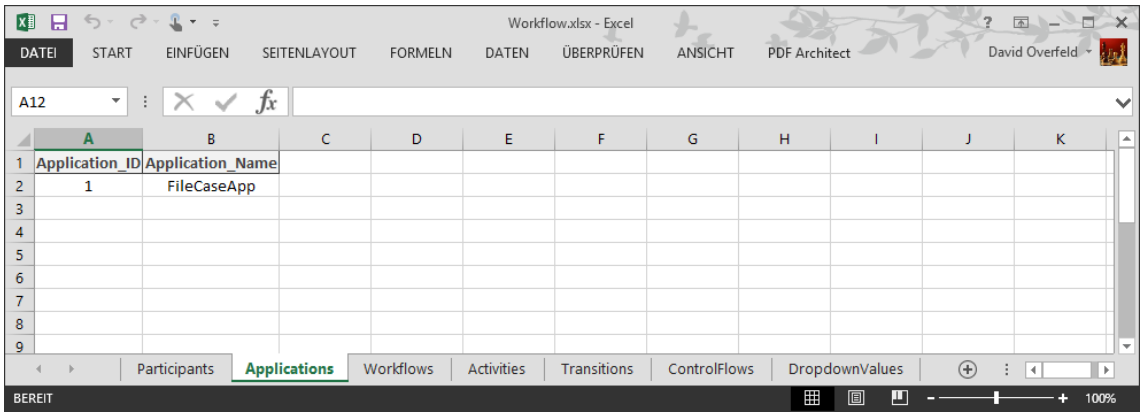


Figure E-3 Workflow.xlsx – Applications

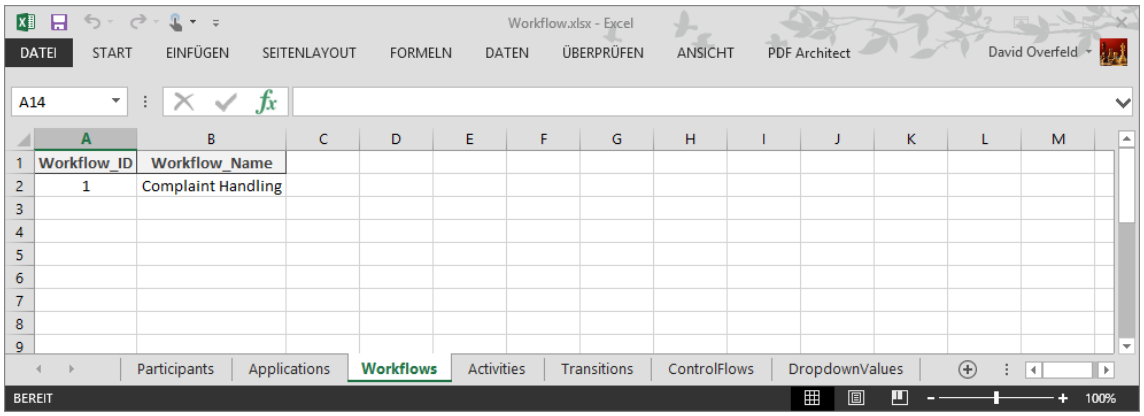


Figure E-4 Workflow. xlsx – Workflows

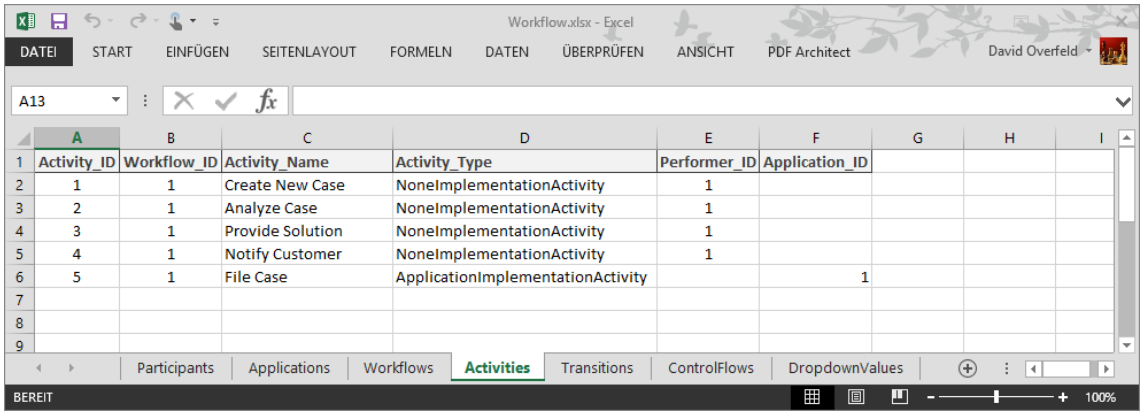


Figure E-5 Workflow.xlsx – Activities

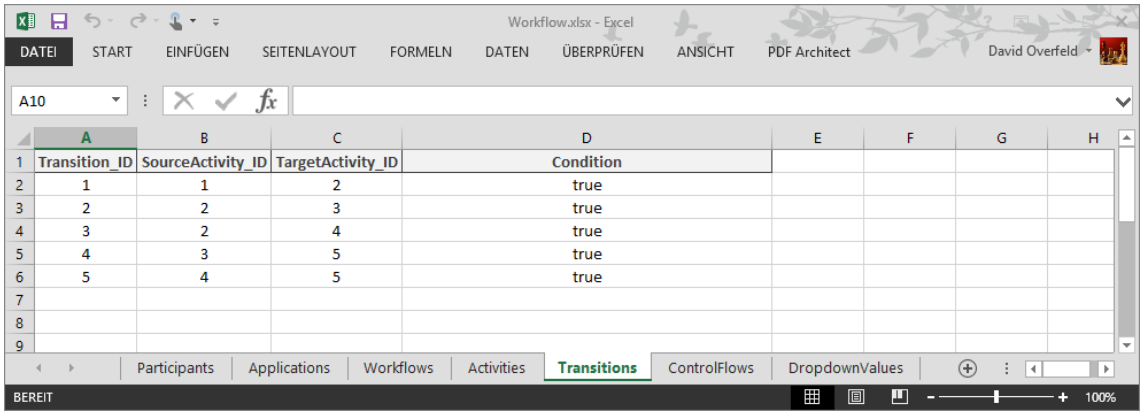


Figure E-6 Workflow.xlsx - Transitions

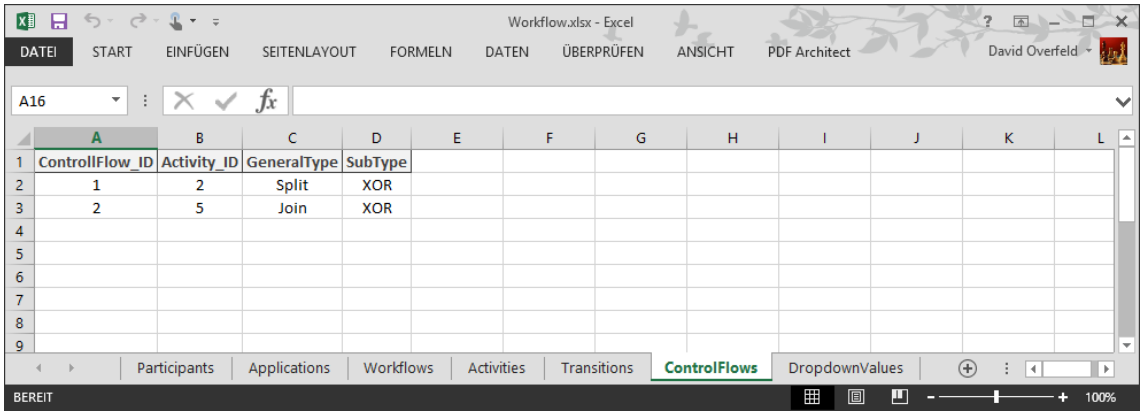


Figure E-7 Workflow.xlsx - Control Flows

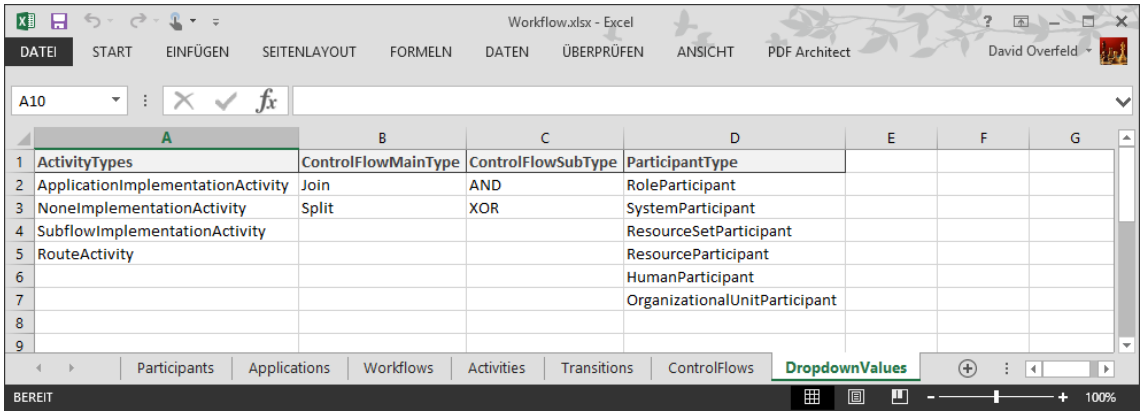


Figure E-8 Workflow.xlsx – Dropdown Values

The following 5 figures depict the content of Event.xlsx.

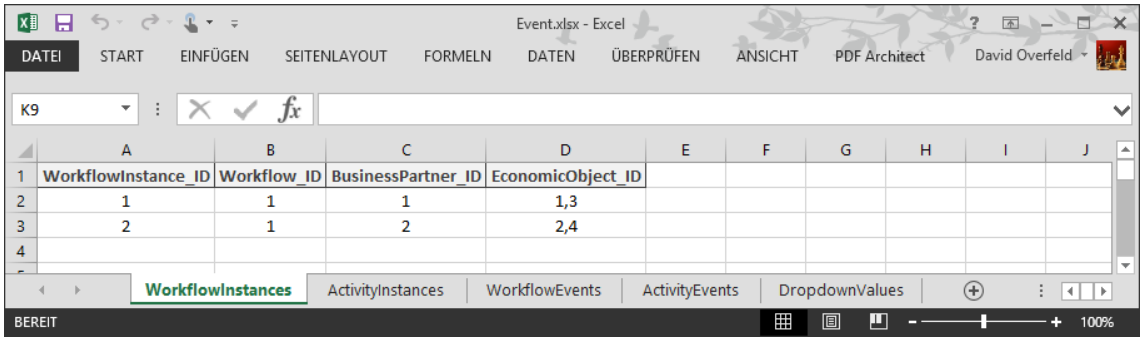


Figure E-9 Event.xlsx – Workflow Instances

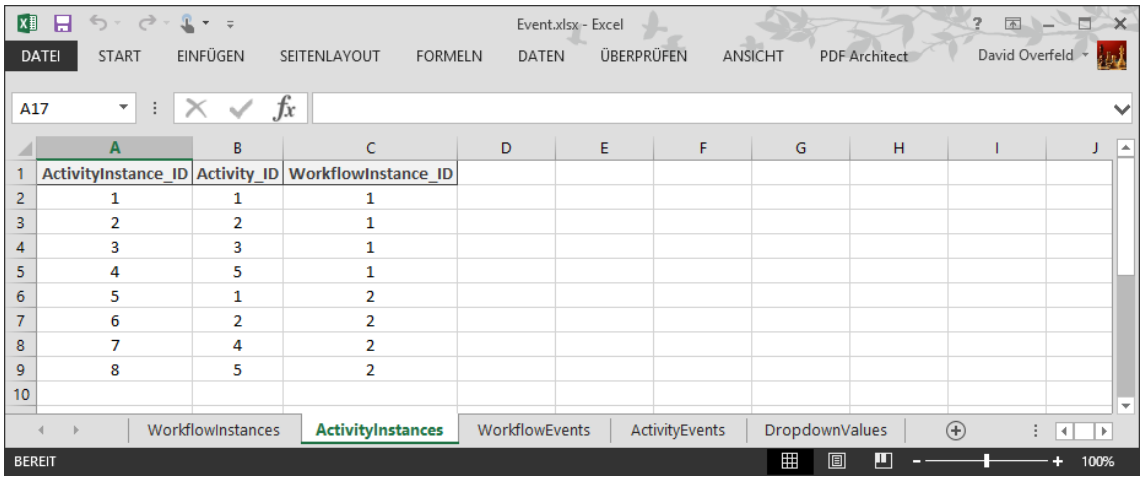
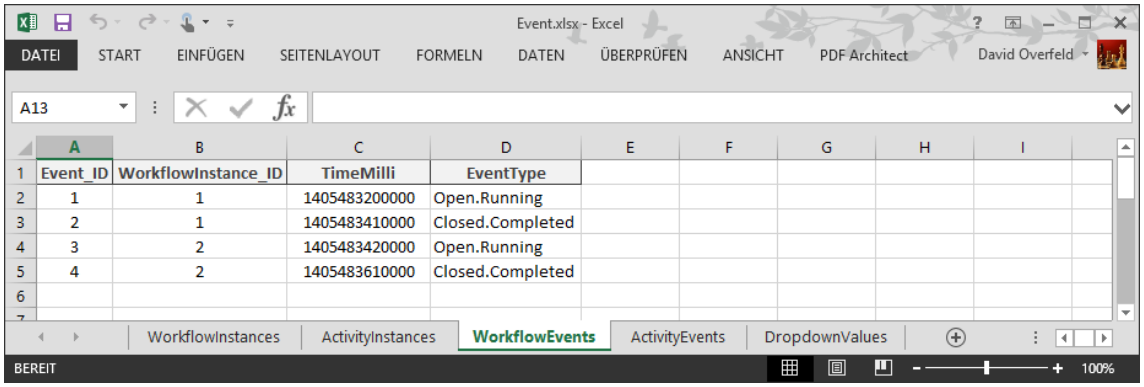
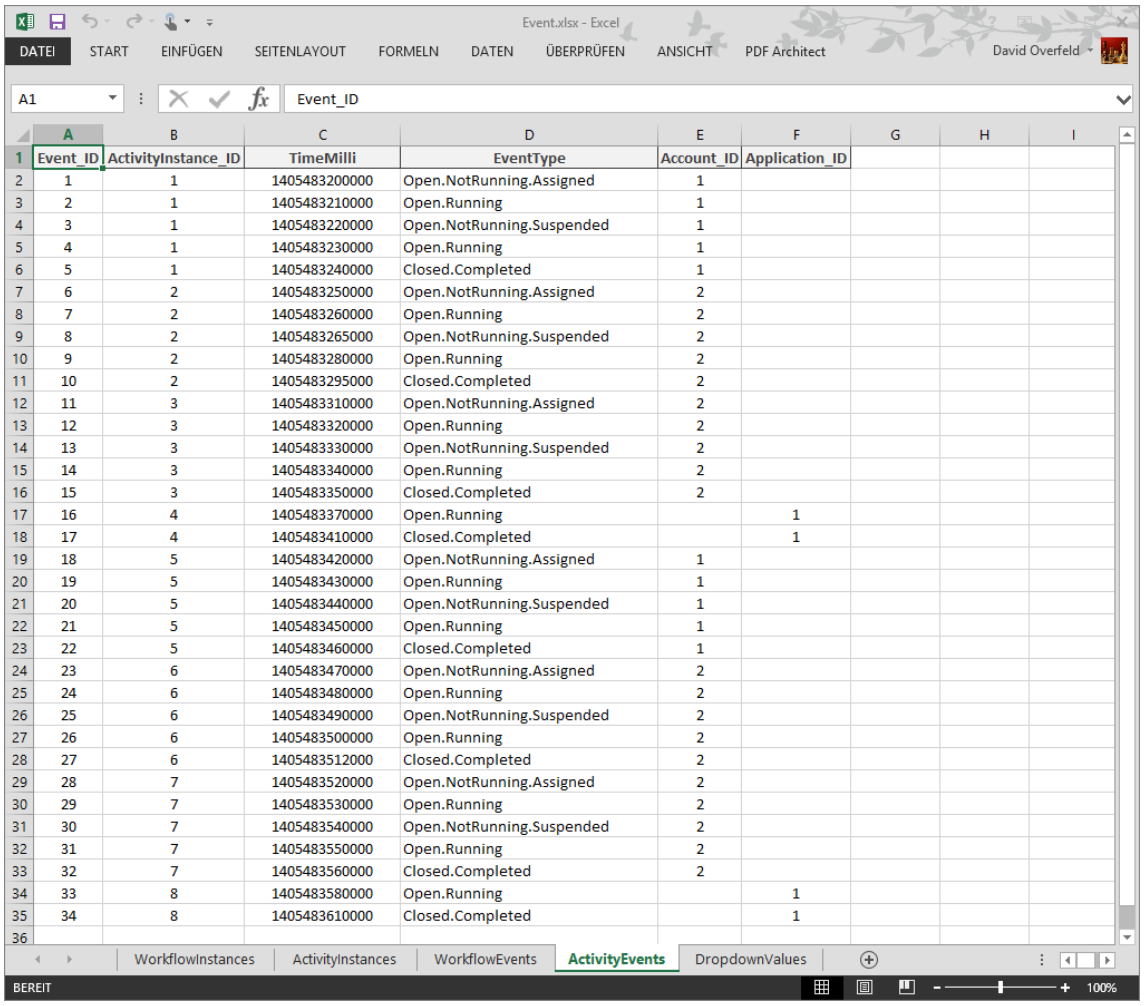


Figure E-10 Event.xlsx – Activity Instances



	A	B	C	D
1	Event_ID	WorkflowInstance_ID	TimeMilli	EventType
2	1	1	1405483200000	Open.Running
3	2	1	1405483410000	Closed.Completed
4	3	2	1405483420000	Open.Running
5	4	2	1405483610000	Closed.Completed
6				

Figure E-11 Event.xlsx – Workflow Events



	A	B	C	D	E	F
1	Event_ID	ActivityInstance_ID	TimeMilli	EventType	Account_ID	Application_ID
2	1	1	1405483200000	Open.NotRunning.Assigned	1	
3	2	1	1405483210000	Open.Running	1	
4	3	1	1405483220000	Open.NotRunning.Suspended	1	
5	4	1	1405483230000	Open.Running	1	
6	5	1	1405483240000	Closed.Completed	1	
7	6	2	1405483250000	Open.NotRunning.Assigned	2	
8	7	2	1405483260000	Open.Running	2	
9	8	2	1405483265000	Open.NotRunning.Suspended	2	
10	9	2	1405483280000	Open.Running	2	
11	10	2	1405483295000	Closed.Completed	2	
12	11	3	1405483310000	Open.NotRunning.Assigned	2	
13	12	3	1405483320000	Open.Running	2	
14	13	3	1405483330000	Open.NotRunning.Suspended	2	
15	14	3	1405483340000	Open.Running	2	
16	15	3	1405483350000	Closed.Completed	2	
17	16	4	1405483370000	Open.Running		1
18	17	4	1405483410000	Closed.Completed		1
19	18	5	1405483420000	Open.NotRunning.Assigned	1	
20	19	5	1405483430000	Open.Running	1	
21	20	5	1405483440000	Open.NotRunning.Suspended	1	
22	21	5	1405483450000	Open.Running	1	
23	22	5	1405483460000	Closed.Completed	1	
24	23	6	1405483470000	Open.NotRunning.Assigned	2	
25	24	6	1405483480000	Open.Running	2	
26	25	6	1405483490000	Open.NotRunning.Suspended	2	
27	26	6	1405483500000	Open.Running	2	
28	27	6	1405483512000	Closed.Completed	2	
29	28	7	1405483520000	Open.NotRunning.Assigned	2	
30	29	7	1405483530000	Open.Running	2	
31	30	7	1405483540000	Open.NotRunning.Suspended	2	
32	31	7	1405483550000	Open.Running	2	
33	32	7	1405483560000	Closed.Completed	2	
34	33	8	1405483580000	Open.Running		1
35	34	8	1405483610000	Closed.Completed		1
36						

Figure E-12 Event.xlsx – Activity Events

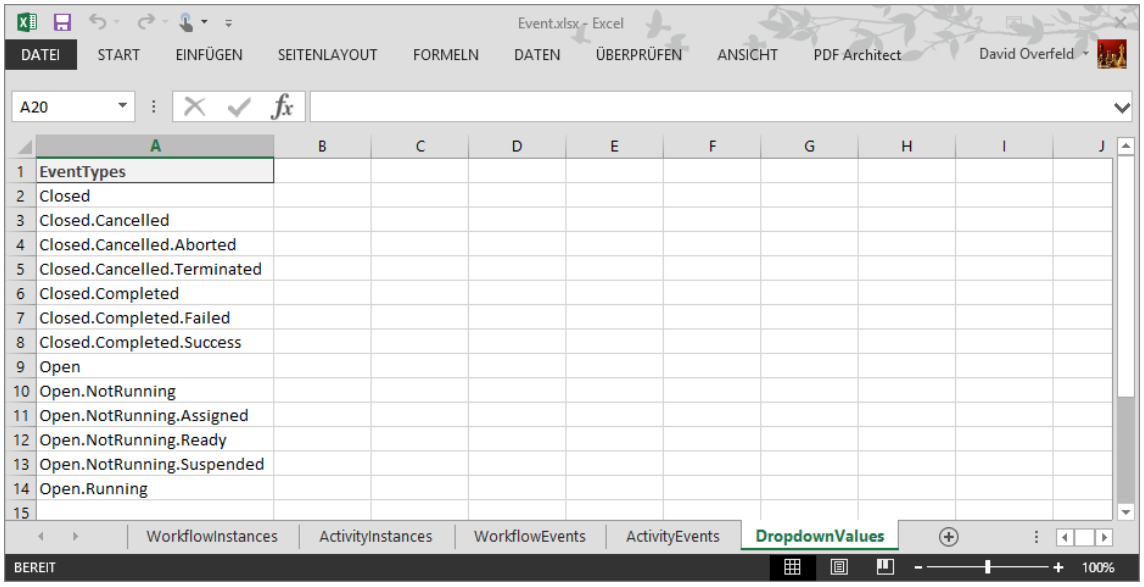


Figure E-13 Event.xlsx – Dropdown Values

The following 4 figures depict the contents of Context.xlsx.

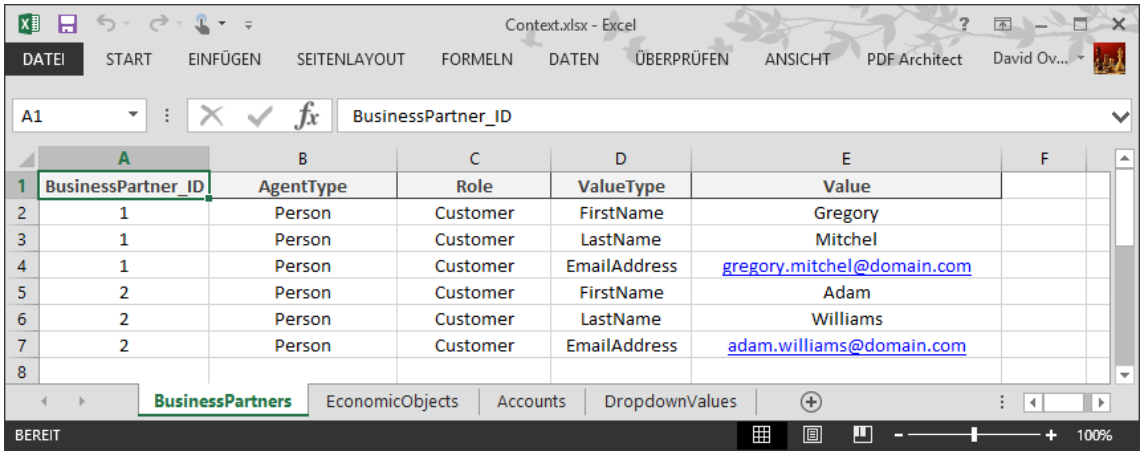


Figure E-14 Context.xlsx – Business Partners

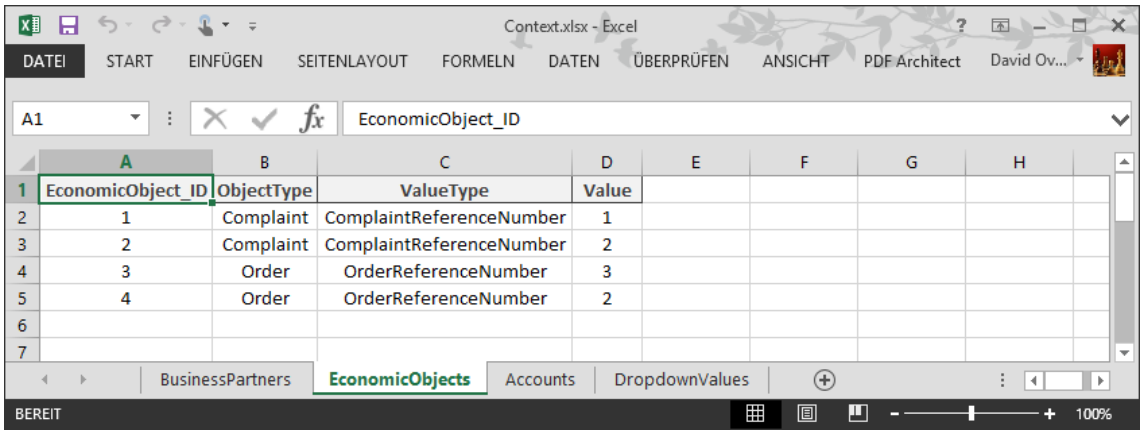


Figure E-15 Context.xlsx – Economic Objects

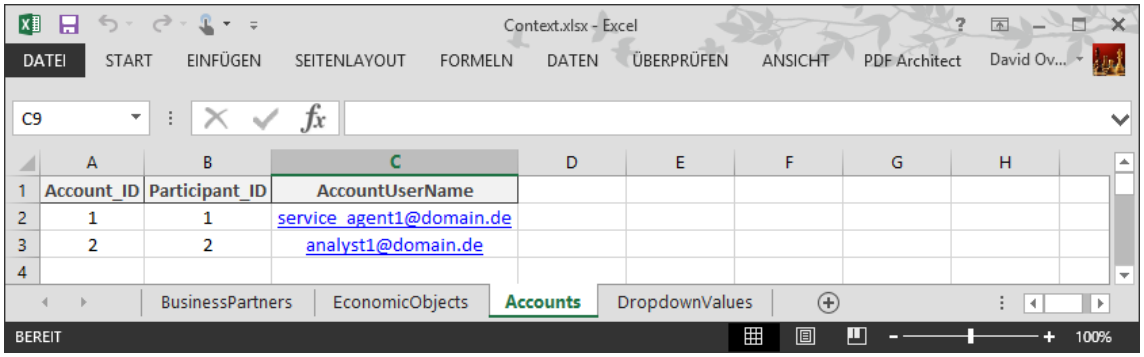


Figure E-16 Context.xlsx –Accounts

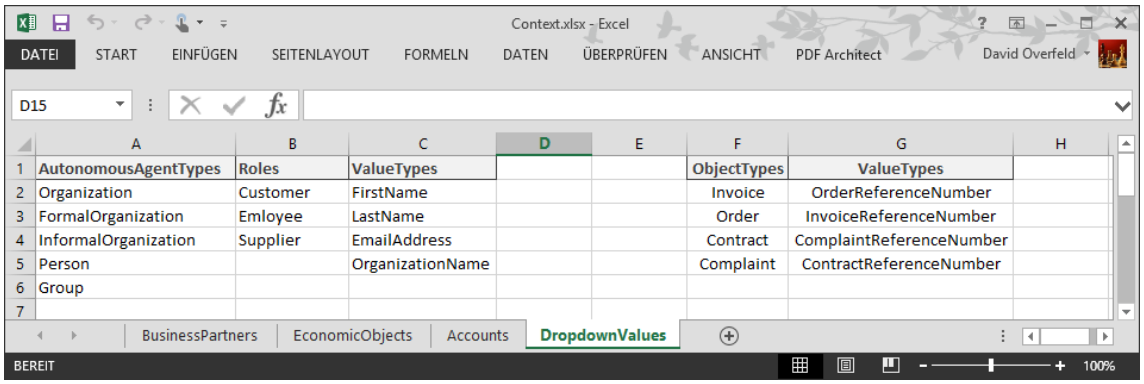


Figure E-17 Context.xlsx – Dropdown Values

F SPA Project Structure (IntelliJ)

To be able to better understand the prototype and its underlying project structure, it is highly recommended to first watch the introductory video provided at http://www.playframework.com/. It is also useful to first create a small test project before working directly with the prototype to better understand the basic principles of the play framework.

How to open the Project

To open the project, first download the newest version of IntelliJ (Community Edition). When IntelliJ is successfully installed, start it and select File/Open and navigate to the prototype's location. Important: Instead of opening the file *project*, open the file *build.sbt*.

Important Files and Folders

Folder *DB_Analysis*: Triple store used for importing and analyzing process event data. If deleted, the prototype will automatically recreate an empty triple store.

Folder *DB_User*: Triple store used for storing internal data used by the SPA. These are user accounts and queries created by the users. If deleted, the prototype will automatically recreate an empty triple store.

Folder *lib*: Contains all external jar-files used by the project. If a new jar-file is added, the IntelliJ-Project has to be restarted. After the restart, the new file will be automatically included into the build path and can be used by program code.

Folder *public*: Contains CSS and JavaScript files as well as images used by the web site. HTML files are not existent since they are integrated into special templates used by the framework.

File *properties*: Contains the ontology namespaces and file locations used by the prototype. Can be manually changed or updated via the prototype's settings page.

Folder views: Contains the HTML pages displayed by the prototype embedded into a scale file, a special template language for dynamically generating web sites.

File *conf/routes*: Connects URLs with functions defined inside the java files *Navigation* and *Application* inside the *controllers* folder.

Folder *controllers*: Contains the java file *Navigation* specifying functions checking the login status and returning views as well as the java file *Application* specifying functions initiating parts of the business logic.

Folder *logic*: Contains the actual logic, including the abstract Importer class, the Model API (ModelCreator) as well as the different importers (SyntheticImporter, StardustImporter, CamundaImporter).

File *public/js/config/config.js*: Contains all SPARQL queries used by the prototype's analysis functions.

G SPA Deployment Manual

The latest version of the prototype can be found in the digital appendix in the folder *SPA Project*. To start and use the prototype locally, the following steps have to be performed.

- In case the prototype had been started before and closed afterwards, delete the file *RUNNING_PID* in the root directory (if existent).
- Start the file *activator.bat* in the root directory
- A command window will appear that will automatically open a browser window with the name *Typesafe Activator*.
- A popup window will appear. Chose the option to run the application.
- The application will be first compiled and then deployed automatically to an integrated web server.
- The prototype is now reachable via the address http://localhost:9000.

If the same steps were performed on a web server that allows access to port 9000, the prototype would be now available under http://$websever$:9000, whereby $webserver$ has to be replaced with the URL the web server can be reached by from the outside.

H Jena Fuseki Manual

Jena Fuseki is a component of the Apache Jena Semantic Web Framework and can downloaded at http://jena.apache.org/download/index.cgi. Jena Fuseki does not have to be installed and can be started via the command line. The following commands have to be executed to start a new Fuseki instance and to insert RDF data into the internal data store:

- Open a new command line window.
- Navigate to the folder where Fuseki was downloaded to. For instance execute the following two commands:
    - *D:*
    - *cd D:\Prototype\jena-fuseki-1.0.1-distribution\jena-fuseki-1.0.1*
- Start a new Fuseki instance by executing the following command:
    - *fuseki-server --update --mem /ds*
- Open the browser and enter the following address:
    - http://localhost:3030
- Select *Control Panel* under *Server Management.*
- Select dataset *ds* and press *Select.*
- In the file upload section, press *Choose File* and select the RDF file whose triples should be uploaded to the data store.
- Press *upload.*
- After the import is finished, press *Back to Fuseki.*
- The SPARQL endpoint can now be queried against the uploaded data via http requests or via the user interface. To perform a SPARQL query via the interface, select again *Control Panel* and the press *Select*. Insert a query into the *SPARQL Query* field and press execute.

I Stardust and Camunda Deployment Manual

In the following, the necessary steps to deploy the developed workflows and to import event data from the two systems are explained.

Stardust

1. Open the folder *Real Life Test/Stardust* in the digital appendix. It contains the *Eclipse* folder, preconfigured with Stardust BPM, and the *Workspace* containing the Stardust project as well as the Apache Tomcat server on which the workflow has to be deployed.

2. Start *eclipse.exe* inside the *Eclipse* folder.

3. Select *File/Switch Workspace* and *Other* and choose the workspace contained in the *Stardust* folder.

4. Open the *StardustProject* inside the *Project Explorer*. Right click on *ComplaintHandling.xpdl* and select *Open With/Process Model Editor*.

5. Open the *Servers* view below the modeling area. Remove the existing *StardustProject* from the server by right clicking on it and selecting *Remove*.

6. Right click on the server and select *Add and Remove*. Add the *StardustProject* from the available projects and click *Finish*.

7. Start the server by right clicking on it and selecting *Start*.

8. In case a new version of the workflow should be deployed, right click on *Stardust Workflows* in the *Outline* on the right side and select *Deploy Model*. Confirm the upcoming dialogs. This step can be skipped when the workflow as presented in this thesis should be used and the existing event data should be imported.

9. The event data can now be imported with the standard settings via the SPA interface.

Camunda

1. Open the folder *Real Life Test/Camunda* in the digital appendix. It contains the *Eclipse* folder, preconfigured with Camunda BPM, the *Workspace* containing the Camunda project as well as the Apache Tomcat server on which the workflow has to be deployed.

2. To deploy the workflow presented in this thesis and to use the existing event data, open the *Server* folder and double click on *start-camunda.bat*. The event data can now be imported with the standard settings via the SPA interface. To redeploy an adapted version of the workflow, the following steps have to be followed.

3. Start *eclipse.exe* inside the *Eclipse* folder.

4. Select *File/Switch Workspace* and *Other* and choose the workspace contained in the *Camunda* folder.

5. Open the new-order folder in the Project Explorer View.

6. Navigate to *src/resources* and double click on *order.bpmn*. The workflow model can now be adapted as desired.

7. Right-click on pom.xml and execute *Run As/Maven install*. A new built will be created.

8. Once the built is finished, open the folder target inside the project folder and copy the contained .war-file.

9. Navigate to Server\server\apache-tomcat-7.0.50\webapps inside the Camunda folder, delete any existing .war-file and add the copied one.

10. Double-click on *start-camunda.bat* inside the *Server* folder. If a new workflow version was added, it will be automatically deployed.

Declaration of Authorship

I hereby declare that, to the best of my knowledge and belief, this Master Thesis titled “A Prototype for Semantic Process Intelligence” is my own work. I confirm that each significant contribution to and quotation in this thesis that originates from the work or works of others is indicated by proper use of citation and references.

Hoboken, 20 August 2014