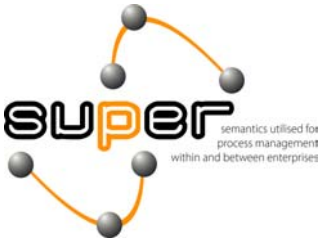


Project IST 026850 SUPER

Semantics Utilized for Process management within and between Enterprises

Deliverable 6.11

Semantic Process Mining Tool – Final Implementation

Leading Partner: TUE

Contributing Partner: NUIG, OU

Security Classification: Public (PU)

September, 2008

Version 1.0

Project Details

IST Project Number	026850
Acronym	SUPER
Project Title	Semantics Utilised for Process management within and between EnteRprises
Project URL	http://www.ip-super.org
EU Project Officer	Werner Janusch

Authors (Partner)	Ana Karla Alves de Medeiros (TUE), Peter van den Brand (TUE), Wil van der Aalst (TUE), Ton Weijters (TUE), Walid Gaaloul (NUIG), Carlos Pedrinaci (OU)		
Deliverable Owner (Partner)	Ana Karla Alves de Medeiros (TUE)	**E-mail**	a.k.medeiros@tue.nl
		Phone	+31 40 247 4239

Versioning and Contribution History

Version	Description	Comments
0.1	Definition of Outline	All partners
0.2	Added text to the sections Executive Summary, Introduction, Ontology Discovery, and Converter from Execution History to SA-MXML	TUE
0.3	Added text to the section Enriching Logs with Semantic Information	NUIG
0.4	Added remaining sections plus corrections based on the comments of our internal reviewer (Irene Celino, from CEFRIEL)	TUE
0.5	Added final text to the section Enriching Logs with Semantic Information	NUIG
1.0	Final editing of the document (based on EPBM delegate feedback)	TUE

Table of Contents

Table of Figures

Executive Summary

D6.11 aims at developing the final version of the *Semantic Process Mining* tool. This tool supports the analysis of processes based on their execution instances (or process space). Actually, D6.11 is a follow-up of D5.2 (cf. Section 4) [1] and D6.5 [2] (where the first semantic process mining prototype has been implemented). In a nutshell, D6.11 focuses on three aspects that have not been addressed in D5.2 and D6.5:

1. The development of *new semantic performance analysis metrics* and corresponding new plug-ins in ProM. These metrics enhance the current performance analysis that is supported by the *Performance Analysis with Ontologies* (cf. [2], Section 4.6), which is able to compute throughput and execution times for the instances of concepts referenced in the logs.
2. The definition and implementation of techniques that help users *to migrate from non-semantic Business Process Management (BPM) environments to semantic ones*. These techniques aim at (i) discovering role ontologies from scratch (based on non-semantic logs) and (ii) enriching (partially annotated) logs with semantic information. The main difference from these new developments to existing related work is the focus on aiding the annotation of semantic BPM environments.
3. Fully integrating the Semantic Process Mining Tool in the SUPER architecture. This integration has consisted on implementing the ProM*import* plug-in "SUPER Project", which automatically convert the WSML instances in the Execution History Repository (the component in the SUPER architecture that stores data relating to the execution of processes) to the SA-MXML logging format (the semantically annotated format that is used by the ProM tool to perform semantic process mining).

Together, D5.2, D6.5 and D6.11 present the Semantic Process Mining tool that has been developed within SUPER.

1 Introduction

The SUPER project aims at developing an architecture in which business people become more independent of the IT people to perform business process management (BPM). To do so, SUPER will embed semantic information in each of the four phases of the BPM lifecycle: design, configuration, execution and analysis. This deliverable (D6.11) focuses on the analysis phase. The aim is to develop the final version of the semantic process mining techniques that are provided by the semantic analysis environment explained in D5.2 [1]. The semantic process mining techniques allow for the semantic analysis of process instances. Furthermore, the results in this deliverable are additions to the results that were already presented in D6.5 [2], where the prototypical version of the semantic process mining tool is described.

In this section we explain how this deliverable aligns with the goal of the SUPER project (cf. Subsection 1.1), the approach that we have taken while developing the process mining tools (cf. Subsection 1.2) and how the remainder of this document is organized (cf. Subsection 1.3).

1.1 Alignment

The semantic process mining tools are a subcomponent of the semantic analysis environment. Therefore, this section explains how the semantic analysis environment integrates with the SUPER architecture, methodology, modeling stack, ontology stack and use cases.

1.1.1 Architecture Alignment

The SUPER reference architecture contains the core elements depicted in Figure 1. As can be seen, this architecture contains four groups of elements (*execution*, *tooling*, *services*, and *repositories*) that are connected via the *semantic service bus*[1]. The semantic analysis environment is part of the *Analysis Tool* in the *SUPER tooling* group, and it interacts with the SUPER repositories *Execution History*. The Execution History repository contains the process space, i.e. the executed process instances over which the analysis is conducted.

[1] A detailed description of this reference architecture is out of the scope of this deliverable. For more details, the interested reader is referred to D6.1 [18] and D7.2 [14].

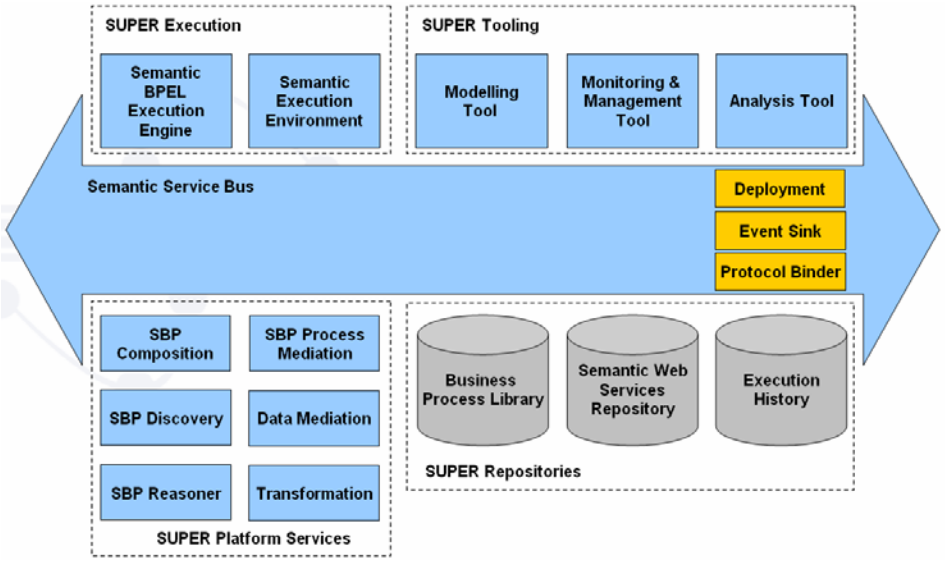


Figure 1 Super Reference Architecture.

1.1.2 Methodology Alignment

The SUPER methodology, like the usual BPM lifecycle, contains four phases (cf. Figure 2): *semantic business process modeling, semantic business process configuration, semantic business process execution* and *semantic business process analysis*. As the name already suggests, this deliverable belongs to the semantic business process analysis phase. The feedback provided by the semantic process mining prototype will aid the (re-)design of processes in the semantic business process modeling phase and their (re-)configuration during the semantic business process configuration phase.

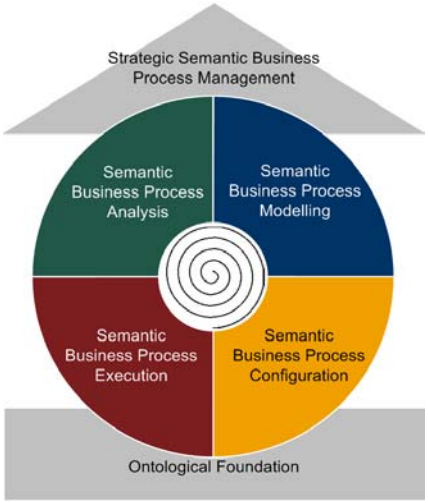


Figure 2 Super Methodology.

1.1.3 Modeling Stack Alignment

The SUPER modeling stack has five layers that follow a top-down approach (cf. Figure 3). The semantic analysis environment aids the second layer, where the process models are designed. The idea is that the semantic feedback provided by the analysis tools will help on detecting points of improvements for existing processes and, therefore, helping on the re-design and/or tuning of these processes.

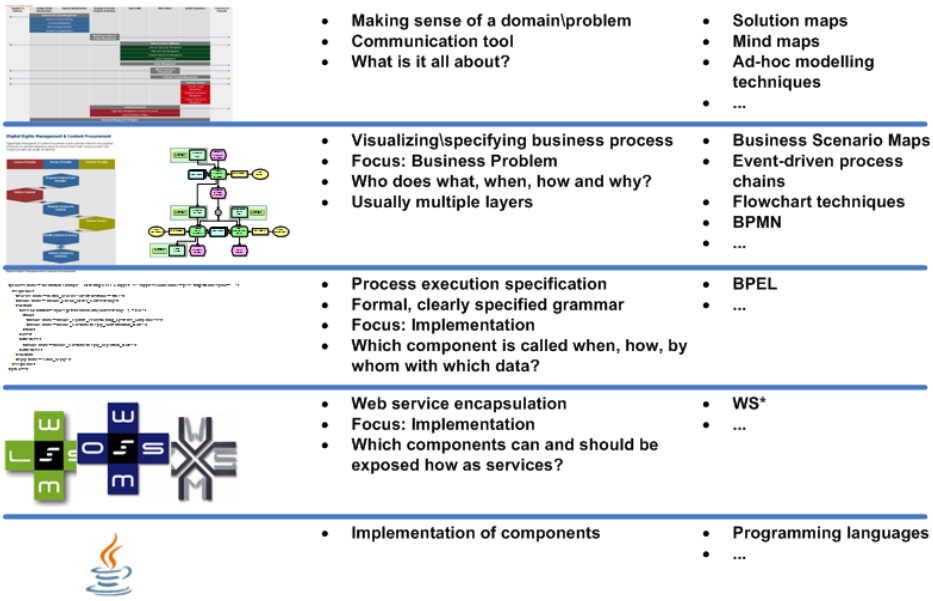


Figure 3 Super Modelling Stack.

1.1.4 SUPER Ontology Stack

Ontologies are the basic building blocks in any semantic analysis. Figure 4 illustrates the SUPER ontology stack (see D1.1 [9] for more details). From this stack, the *Core Business Analysis* (COBRA) ontology is especially important for semantic analysis. The Appendix of this document and deliverable D1.9 [10] contain more details on the COBRA ontology and its related ontologies.

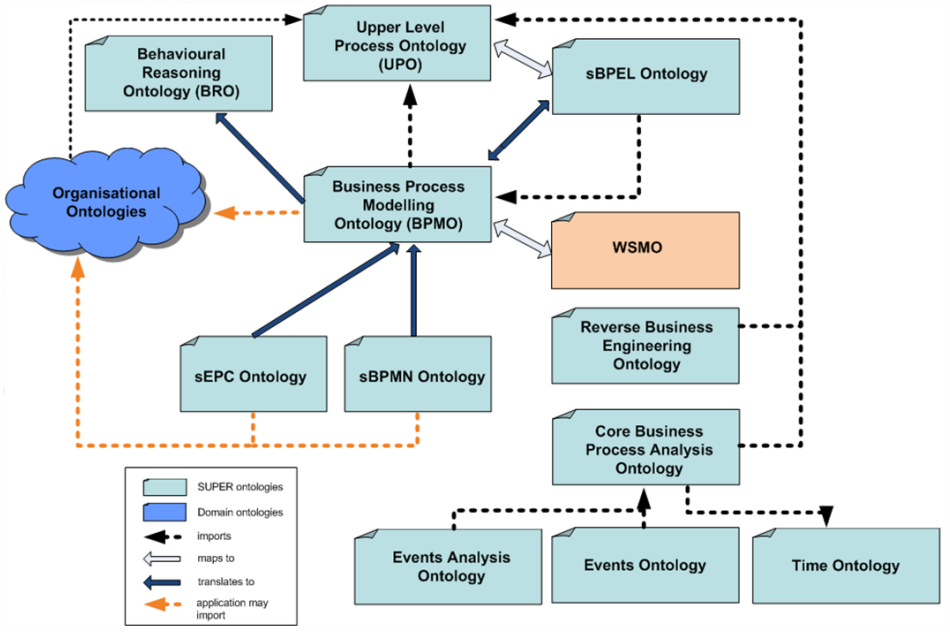


Figure 4 SUPER Ontology Stack.

1.1.5 Use case Alignment

As a proof of concept, the analysis tools developed in the semantic analysis environment are going to be used within the use cases in SUPER. For instances, one could use the semantic process mining tools to analyze the use case "Requesting Service Level Agreement" from the partner Nexcom Telecommunications (see D9.1 [15], section 2.2.5.2 for more details).

1.2 Approach and Results

Process mining [3] targets the automatic discovery of information from *event logs*. The discovered information is used to analyze how the systems that generate these logs are actually being used. The analysis provided by current process mining techniques can be seen as from three types [3]: *discovery*, *conformance* and *extensions* (cf. Figure 5). The techniques that focus on *discovery* mine information based on data in an event log only. This means that these techniques do not assume the existence of pre-defined models to describe aspect of processes in the organization. Examples of such techniques are control-flow mining algorithms that extract a process model based on the dependency relations that can be inferred among the tasks in the log. Other examples are social network mining algorithms that discover the relations between the performers of certain tasks, like a graph that shows who is handing over work to whom. The algorithms for *conformance* verify if logs follow *prescribed* behaviors and/or rules. Therefore, besides a log, such algorithms also receive as input a model that captures the desired property or behavior to check. Example of such algorithms are the mining algorithms that assess how much the behavior expressed in a log matches the behavior defined in a model and points out the differences, or algorithms used for auditing of logs (in this case, the model is the property to be verified). The *extension* algorithms enhance existing models based on information discovered in event logs, e.g., algorithms that automatically discover business rules for the choices in a given model.

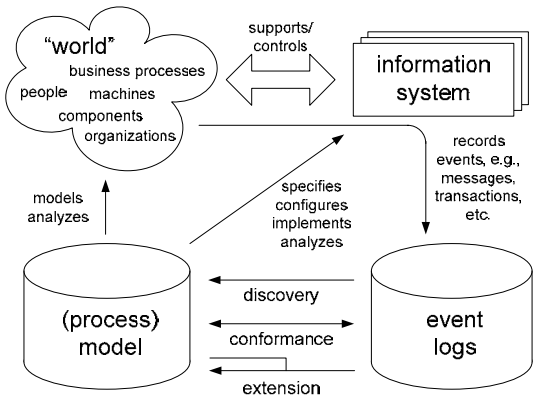


Figure 5 Sources of information for process mining. The discovery plug-ins use only an event log as input, while the *conformance* and *extension* plug-ins also need a (process) model as input.

Current discovery, conformance and extension process mining techniques are already quite powerful and mature. However, the analysis they provide is purely syntactic. In other words, these mining techniques *are unable to reason over the concepts behind the labels in the log*, thus the actual semantics behind these labels remain in the head of the business analyst which has to interpret them. Therefore, within SUPER we are developing process mining techniques that make use of this semantic perspective.

In fact, the semantic perspective provided by SUPER raises two opportunities [5] for process mining techniques. The *first opportunity* is to make use of the ontological annotations in logs/models to develop more robust process mining techniques that analyze the logs/models at the concept level. In this case, it is assumed that event logs and models indeed link to ontologies. The *second opportunity* is to use process mining techniques to discover or enhance ontologies based on the data in event logs. The semantic process mining prototype presented in the D6.5 [2] has exploited the first opportunity. The final version of the semantic process mining tools (presented in this deliverable) also addresses the second opportunity (i.e., the use of process mining techniques to discover or enhance ontologies).

The semantic process mining prototype is based on the ProM tool [11], which is an open-source framework that supports the development of process mining algorithms/techniques in a pluggable way. *For the deliverables D5.2* [1] *and D6.5* [2], seven semantic plug-ins have been implemented: (i) *Ontology Summary*, which shows a projected view of an ontology; (ii) *Semantic LTL Checker* [4], which supports the verification of properties in a log, e.g. for auditing purposes; (iii) *Semantic Control-Flow Mining*, which allows for the mining of process models at different levels of abstractions; (iv) *Semantic Performance Analysis*, which shows information about throughput times of process instances and processing times of tasks based on concepts in a log; (v) *Semantic Organizational Mining*, which uses the semantic information in the log to mine groups of users based on task similarity; (vi) *Semantic Originator by Tasks Matrix*, which describes how frequently certain user concepts execute certain task concepts; and (vii) *Semantic Ontology URI Renaming Filter*, which supports renaming the location of ontologies used in a log. *For the current deliverable (D6.11), four other semantic ProM plug-ins have been implemented:* (i) *Role Hierarchy Miner*, which mines the backbone of an organizational ontology based on execution instances of a given process; (ii) *Annotate with Default Ontologies*, which allows end users to apply the semantic process mining plug-ins over non-semantic logs; (iii) *Semantic Log Enricher*, which automatically adds semantic annotations to elements in events logs; (iii) *Execution Times Using Availability Based on Hours Per Shift*, which calculates the processing (or execution) times of tasks by considering the time shifts in which performers where available. This analysis includes more sophisticated metrics that go beyond the analysis already supported by the plug-in *Performance Analysis based on Ontologies* (cf. Section 4.6 in [2]).

All semantic plug-ins are part of the subcomponent semantic process mining (sPM) of the semantic analysis environment (cf. Figure 6) and are publicly available in the ProM tool[2]. Note that the sPM

[2] ProM 5.0 [10] contains all the semantic plug-ins described in D5.2 and D6.5, plus the semantic plug-

component converts the information in the execution history to the SA-MXML (Semantically Annotated Mining XML format) [2]. Therefore, *within the context of D6.11 we have also developed the actual component to convert the information in the Execution History repository to the SA-MXML format.* This format is our semantic extension to the log input format used by ProM. Furthermore, the SA-MXML format is backwards compatible with the Mining XML (MXML) [13] format used by the non-semantic plug-ins of the ProM tool. Therefore, users of our sPM component can directly utilize other non-semantic process mining plug-ins that are provided in ProM. As a consequence, these users have an even bigger myriad of process mining techniques to analyze process instances.

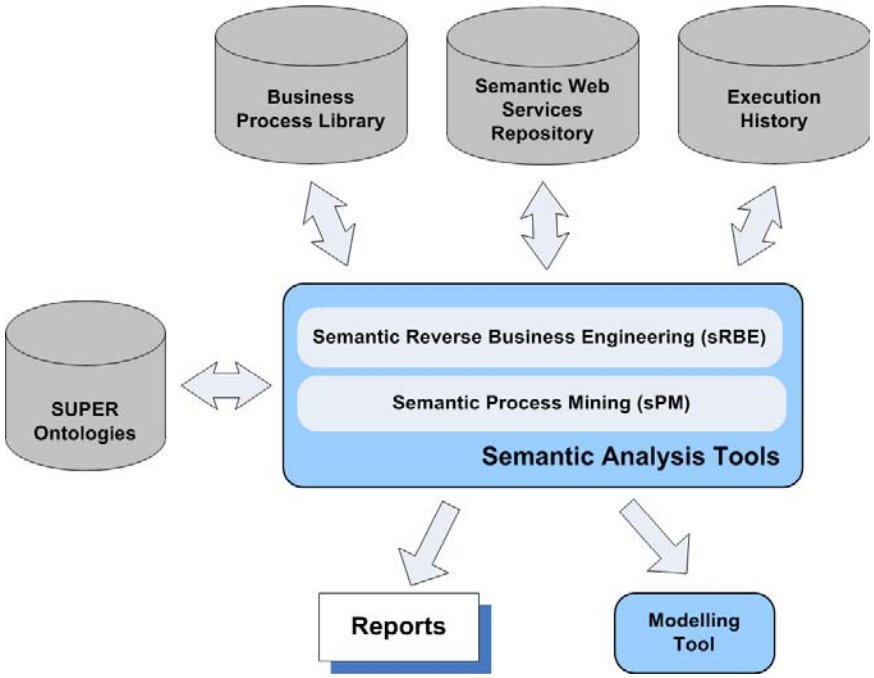


Figure 6 Semantic Analysis Tools in SUPER.

ins *Role Hierarchy Miner* and *Annotate with default ontologies* (cf. Section 3). The other semantic plug-ins described in D6.11 are publicly available together with the nightly builds of the ProM tool (see http://tabu.tm.tue.nl/dev/prom/nightly/). This is the case because these other plug-ins described in have been developed after the official release of ProM 5.0. Therefore, these plug-ins will be incorporated in the next official release of the ProM tool.

1.3 Document Structure

The remainder of this document is organized as follows. Section 2 describes how process mining techniques can be used to discover *organizational* ontologies. This section presents also the ProM plug-ins *Role Hierarchy Miner* and *Annotate with Default Ontologies*. Section 3 explains the *Semantic Log Enricher* plug-in, which adds semantic annotation to event log. Section 4 describes the *Execution Times Using Availability Based on Hours Per Shift* plug-in, which takes into account the availability of performers when computing execution (or processing) times of tasks. Section 5 describes how to use the *SUPER Project* ProM*import*, which is the component to convert the data in the Execution History to the SA-MXML format, and Section 6 contains the conclusions. For the interested reader, Table 1 outlines the roadmap based on the three points explained in the Executive Summary.

Point in the Executive Summary	**Sections with details on these points**
1. New Semantic Performance Metrics	Section 4
2. Migration from non-semantic BPM environment to semantic ones	Section 2 and Section 3
3. Component to create SA-MXML logs in SUPER	Section 5

Table 1 Roadmap based on points in the Executive Summary.

2 Ontology Discovery

Deploying semantic BPM systems (like the SUPER framework) can be very time consuming because several ontologies have to be provided. Therefore, within D6.11, we have developed an approach based on process mining techniques to discover ontologies from event logs. *The focus is on mining templates for role ontologies*. The discovered templates contain concepts, instances and is-a relationships. The term "template" is used because it is up to the end user to give meaningful labels to the automatically discovered concepts. The applicability of the approach is illustrated by showing the ProM plug-ins to mine and export role ontologies from an event log. Both the approach and the developed plug-ins are explained in Subsection 2.1. Additionally, in order to allow users to quickly apply the semantic process mining ProM plug-ins over non-semantic logs too, we have developed a simple algorithm to *annotate logs with default ontologies* that are created based on the MXML [13] logging format. Subsection 2.2 contains more details about this algorithm and its implementation in ProM. All the plug-ins described in this section have been officially released together with ProM 5.0 [11]. Furthermore, all the screenshots included in this section are based on a synthetic log that is publicly available at http://is.tm.tue.nl/research/processmining/OntologyMiningLog.zip.

2.1 Mining Role Ontologies: Approach, Algorithm and Implementation

When designing our approach, we had the following two main requirements as input: (i) the mining of role ontologies should be based on *actual executions* of business processes; and (ii) the approach should support the easy replacement of involved components. This facilitates the (re-)use of existing or future solutions. An overview of the resulting approach is illustrated in Figure 7. Note that, satisfying requirement (i), *event logs* are the starting point of our approach. Furthermore, satisfying requirement (ii), the components (or modules) in the approach is structured in the piped way, where each component has a well-defined interface and can be easily substituted. The whole approach is as follows: Based on event logs, process mining techniques can be used to mine an *originator by task matrix*. The *matrix* is the input for the algorithm to mine the populated backbone template of a role ontology. This algorithm is called `RoleOntologyMiner`. In a nutshell, this algorithm uses the information about which originators have executed which tasks to identify who are the specialists/generalists [12] for a given process. Subsection 2.1.1 contains more details about the `RoleOntologyMiner` algorithm. Subsection 2.1.2 introduces the ProM plug-ins that have been developed to mine and export role ontologies. This subsection also explains how to use these plug-ins.

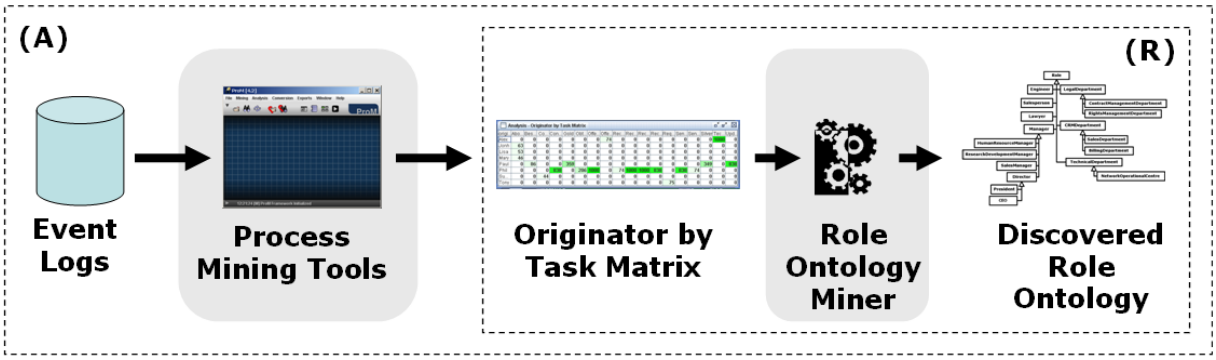


Figure 7 Illustration of our approach. The whole approach in (A) starts with event logs and, by applying the ontology mining algorithms explained in this section, can discover populated templates for role ontologies. The area in (R) indicates the input (i.e., the "Originator by Task Matrix") and output (i.e., the "Discovered Role Ontology") of the algorithm RoleOntologyMiner. The grey rectangles highlight the two components involved in the approach.

2.1.1 Role Ontology Miner Algorithm

In an organization, roles can be used to set the permissions that users and systems have to perform certain tasks. For instance, think of roles like secretary, manager, team leader, director, etc. These roles provide an abstraction layer to detach the execution of tasks from the actual people or systems by focusing on the required capabilities or competencies necessary to perform these tasks. These competencies naturally capture the notion of specialists and generalists in organizations. Specialists can execute a more restricted set of tasks than generalists [12]. In other words, *the set of tasks that a generalist can execute is a superset of the set of tasks that a specialist can*. This is the basic notion behind the `RoleOntologyMiner` algorithm formalized in Definition 1.

The `RoleOntologyMiner` algorithm works as follows. First, it creates a concept for each set c containing *all* the originators that perform the same tasks (Steps 1 and 2). Second, it defines the instances of these concepts (Step 3). Each originator is an instance of the concept it belongs to (Step 4). Third, it builds the is-a relationships for the concepts in C (Steps 5 and 6). A concept c_1 is a subconcept of another concept c_2 if all the originators in c_1 can execute the tasks that the originators in c_2 can execute. Because some concepts may not have a superconcept, an artificial *root* node is created in Step 7, and new is-a relationships are established between these concepts and the root node (Step 8). The discovered ontology is returned in Step 9.

Definition 1. (Mining Algorithm `RoleOntologyMiner`) *Let* $M : O \times T \rightarrow \mathbb{N}$ *be the matrix that describes how often an originator* $o \in O$ *has performed a task* $t \in T$. *Let* $\texttt{perform}_M : O \rightarrow \mathcal{P}(T)$ [1] *be a function that returns the set of tasks that an originator has performed, i.e.,* $\texttt{perform}_M(o) = \{t \in T | M(o,t) > 0\}$. $\texttt{RoleOntologyMiner}(M)$ *is defined as follows.*

1. $C' = \{c \subseteq O \mid \forall_{o_1,o_2 \in c}\ \texttt{perform}_M(o_1) = \texttt{perform}_M(o_2)\}$
2. $C = \{c \in C' \mid \nexists_{c' \in C'} c \subset c'\}$
3. $I = O$
4. $iOf = \{(o,c) \in O \times C \mid o \in c\}$
5. $\texttt{perform}_M(c) = \bigcup_{o \in c} \texttt{perform}_M(o)$, *for* $c \in C$
6. $isA' = \{(c_1,c_2) \in C \times C \mid \texttt{perform}_M(c_2) \subset \texttt{perform}_M(c_1) \wedge (\nexists_{c_3 \in C} \texttt{perform}_M(c_2) \subset \texttt{perform}_M(c_3) \wedge \texttt{perform}_M(c_3) \subset \texttt{perform}_M(c_1))\}$
7. $Root = \{\{anybody\}\}$
8. $isA = isA' \cup \{(c,r) \in C \times Root \mid \nexists_{c' \in C}(c,c') \in isA'_C\}$
9. $\texttt{RoleOntologyMiner}(M) = (C \cup Root, I, isA \cup iOf)$

[1] $\mathcal{P}(A)$ denotes the powerset of some set A.

The ontology template returned by the `RoleOntologyMiner` algorithm is an objective starting point to build the role ontology for a given process. Actually, this template has more than the backbone of this role ontology because it also contains information about which instances belong to which concepts. Although it is clear that end users will have to improve this ontology (e.g., by giving meaningful names to concepts, defining properties and axioms, etc.), the core structure of the ontology reflects the specialist/generalist relations observed in practice. However, the `RoleOntologyMiner` algorithm in Definition 1 has one drawback: *it does not take the frequencies of the executions into account*. So, if an originator performs a given task at least once, this is enough to state that this originator is able to perform this task. In many situations, it may be desirable to build ontologies that express what the originators have more frequently performed. Note that the more often an originator performs a task, the better (or the more specialized) this originator becomes in doing so. For this reason, we have defined three new functions that can be used as a *replacement* to the function `perform` in Definition 1. These functions are `absolute`, `relative` and `combined`, and they are formalized in Definition 2. The function `absolute` calculates the set of tasks that an originator has performed more than n times. The function `relative` computes the set of tasks that an originator has performed more than a certain percentage m, which is relative to the total number of times this originator has executed tasks. The function `combined`, as the name already suggests, combines the two functions `absolute` and `relative`.

Definition 2. (Replacement Functions) *Let $M : O \times T \rightarrow \mathbb{N}$ be the matrix that describes how often an originator $o \in O$ has performed a task $t \in T$. Let $n, m \in \{0..100\}$ be thresholds. The functions* $\texttt{absolute}_{M,n} : O \rightarrow \mathcal{P}(T)$, $\texttt{relative}_{M,m} : O \rightarrow \mathcal{P}(T)$ *and* $\texttt{combined}_{M,n,m} : O \rightarrow \mathcal{P}(T)$ *are defined as follows.*

- $\texttt{absolute}_{M,n}(o) = \{t \in T | M(o,t) > n\}$
- $\texttt{relative}_{M,m}(o) = \{t \in T | M(o,t) > ((m/100) * \sum_{t' \in T} M(o,t'))\}$
- $\texttt{combined}_{M,n,m}(o) = \texttt{absolute}(o,t,M,n) \cap \texttt{relative}(o,t,M,m)$

Notice that when n equals zero, the function `absolute` in Definition 2 behaves just like the function `performs` in Definition 1. Furthermore, the use of these replacement functions indeed allows the end user to get a more robust feedback about how specialized users actually are. In fact, our ProM plug-in implementation explained in Subsection 2.1.2 supports all these flavours of the `RoleOntologyMiner` algorithm.

2.1.2 Implementation Using ProM and WSML

Two ProM plug-ins have been implemented to *mine* and *export role ontologies*. Figure 8 shows a screenshot with these plug-ins.

The *Role Hierarchy Miner* mining plug-in implements the algorithm `RoleOntologyMiner` (cf. Definition 1). The *middle panel* shows the mined backbone. The squares represent the discovered concepts and the directed arrows represent the mined is-a relationships. Furthermore, by enabling the option "Show tasks" (see left panel), it is possible to see which tasks have been performed by originators in a given concept. This information is shown in the ellipses attached to concepts. For instance, the originators "Lucas" and "Rose" have executed the task "Prepare Silver". Note that the ellipses attached to the concepts only show the tasks that do not belong to any superconcepts. For instance, the concept named "Joan" does not have any ellipsis attached to it because the originator "Joan" has executed only the task "Prepare Best Effort" and "Prepare Silver", which are already linked to the two superconcepts of the concept that contains "Joan" (namely, the superconcept with the user "Carlos" and the superconcept with the users "Lucas" and "Rose"). The *thresholds* "Absolute" and "Relative" (cf. left panel) respectively correspond to the thresholds n and m in Definition 2. Note that setting (one of) these thresholds to values higher than zero is equivalent to replacing the function `perform` in Definition 1 by one of the functions in Definition 2. For instance, if both values are set to zero, the plug-in will behave just like the algorithm in Definition 1. If only the value of "Absolute" is bigger than zero, the plug-in behaves as if the function `perform` is replaced by the function `absolute` in Definition 1. A similar reasoning holds when only the value of "Relative" is bigger than zero, or both thresholds are bigger than zero. The *right panel* allows the end user to zoom in and out on the graph view. The *bottom panel* shows the originator by task matrix after applying the filtering based on the thresholds. Furthermore, if the end user clicks on a concept, this matrix shows only the users that are instances of that concept.

The *Export ontology to WSML* export plug-in (cf. menu option *Exports→Role hierarchy ontology→Export ontology to WSML* in Figure 8) allows for saving the mined role ontology as a WSML ontology [16]. End users can modify exported ontologies in any WSML editor. For instance, Figure 9 shows a view of the exported ontology from Figure 8 in the WSMT (Web Service Modelling Toolkit) editor [17].

How to Use the *Role Hierarchy Miner* Plug-in

To use the *Role Hierarchy Miner*, perform the following steps in the ProM tool:

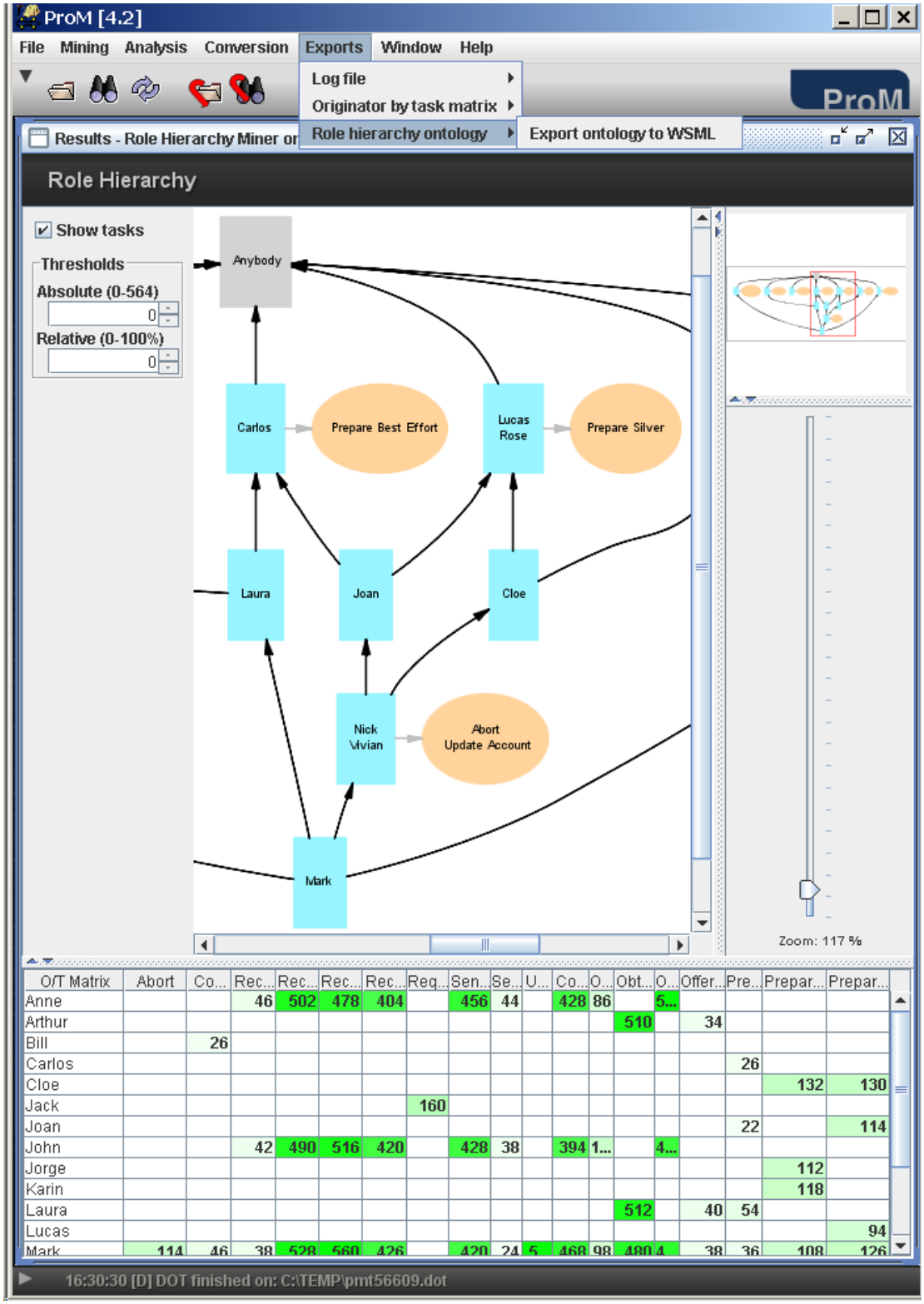
1. Open a (SA-)MXML log file by clicking on *File->Open Supported File....*
2. Start the *Role Hierarchy Miner* plug-in via clicking *Mining->LogName->Role Hierarchy Miner*

How to Use the *Export ontology to WSML* Plug-in

To use the *Export ontology to WSML*, perform the following steps in the ProM tool:

1. Mine a role ontology by using the *Role Hierarchy Miner* plug-in (see above)
2. Export the mined ontology by clicking *Exports->Role hierarchy ontology->Export ontology to WSML*

As a final remark, we point out that our approach can also be used when there is no event log. Although the overall approach in Figure 7 starts with an event log, the algorithm `RoleOntoloyMiner` only requires an originator by task matrix as input (see highlighted area (R) in Figure 7). Therefore, both components highlighted in grey in Figure 7 could be easily replaced by other techniques, with same input and output, that may be developed in the future.



O/T Matrix	Abort	Co...	Rec...	Rec...	Rec...	Rec...	Reg...	Sen...	Se...	U...	Co...	O...	Obt...	O...	Offer...	Pre...	Prepar...	Prepar...
Anne			46	502	478	404		456	44		428	86		5...				
Arthur													510		34			
Bill		26																
Carlos																26		
Cloe																	132	130
Jack							160											
Joan																22		114
John			42	490	516	420		428	38		394	1...		4...				
Jorge																	112	
Karin																	118	
Laura													512		40	54		
Lucas																		94
Mark	114	46	38	528	560	426		420	24	5	468	98	480	4	38	36	108	126

Figure 8 Screenshot of the two ProM plug-ins to mine and save role ontologies. The main window shows the role ontology discovered by the Role Hierarchy Mining plug-in. The highlighted menu option indicates how to invoke the Export ontology to WSML plug-in to save the mined role ontology.

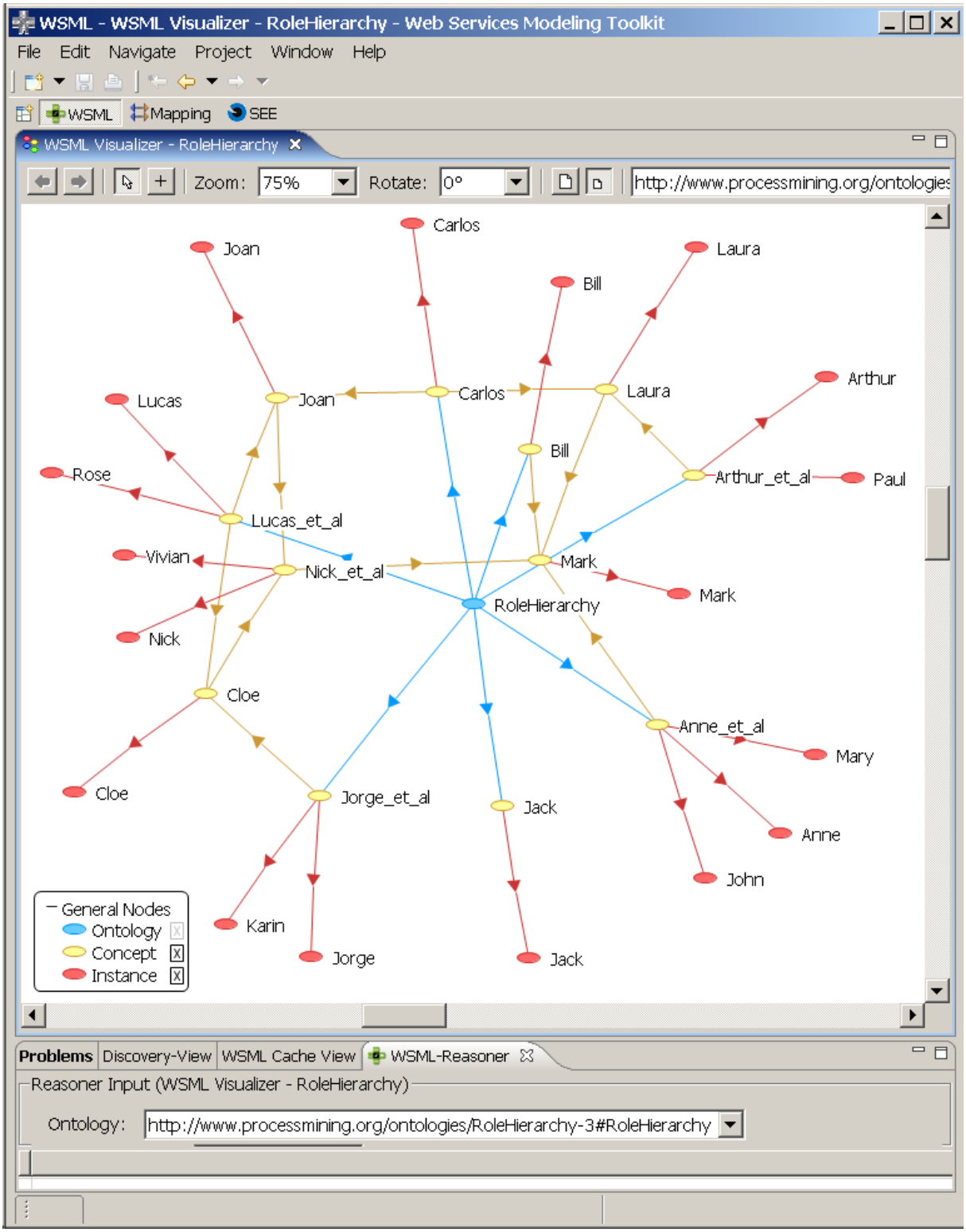


Figure 9 Screenshot of the Role Hierarchy ontology in the WSMT Toolkit, which is an editor for WSML ontologies. Note that, in this case, the end user would have to give meaningful names for the concepts. Our export labels the concepts after the users it contains. When multiple originators belong to a concept, our export adds the substring *_et_al* to the name of the first originator to be an instance of this concept.

2.2 Mining Default Ontologies

The semantic process mining plug-ins require semantically annotated logs (i.e., SA-MXML logs) as input. In this sense, it would be nice if end users could taste the power of the semantic process mining plug-ins without first having to annotate logs. Therefore, within D6.11 we have developed a plug-in that can semantically annotate logs based on the structure of the Mining XML (MXML) [13] format. The created ontologies are called *default ontologies* because they are based on the format of MXML. Actually, because these ontologies are based on this format, they constitute a good starting point to create more elaborate ontologies for a given business process model.

The ProM plug-in that can mine default ontologies is called *Annotate with default ontologies*. Figure 10 shows a screenshot of this plug-in in action. Note that any semantic plug-in can be now called over this log because it has been semantically annotated with default ontologies. As an illustration, Figure 11 contains a screenshot with the results of running the Performance Metrics in Ontologies [2] for this log.

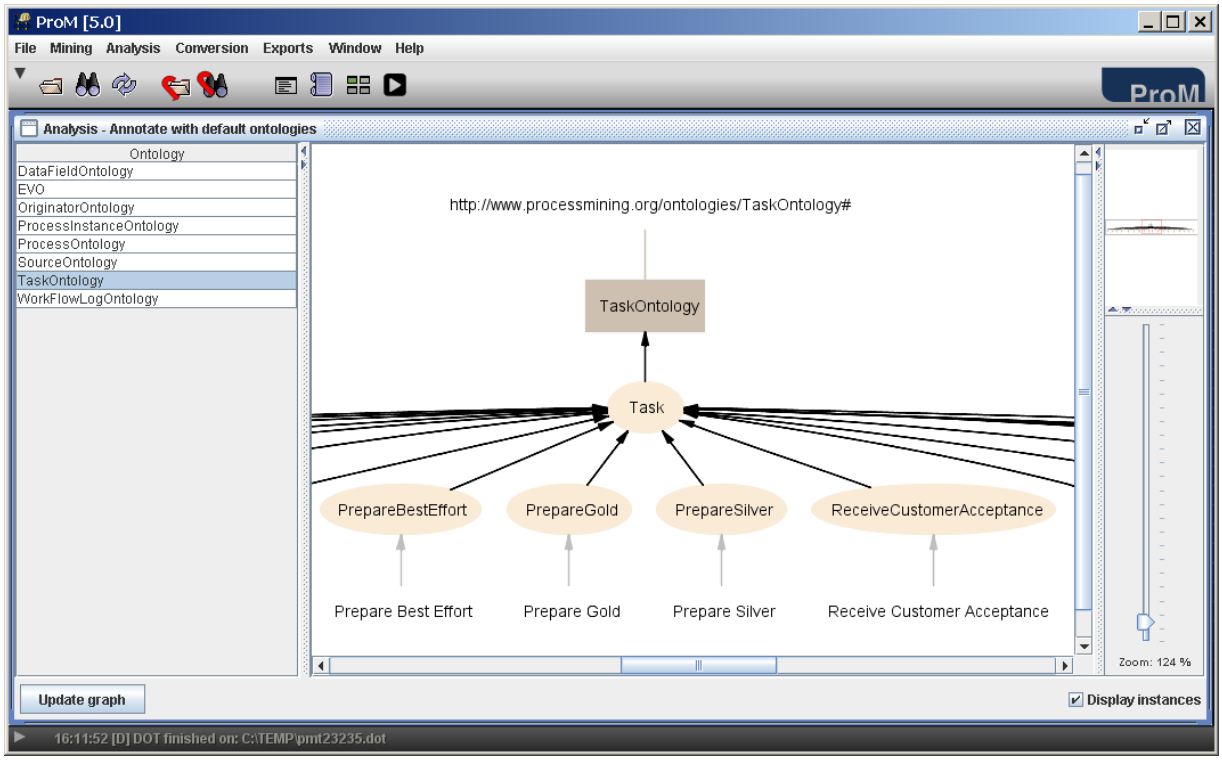


Figure 10 Screenshot showing the results of applying the *Annotate with default ontologies* plug-in to an MXML log.

In total, 7 default ontologies are created and populated based on the elements that are contained in a given log (cf. left pane of Figure 10). These ontologies are: *DataFieldOntology, OriginatorOntology, ProcessInstanceOntology, ProcessOntology, SourceOntology, TaskOntology* and *WorkflowLogOntology*. The *EVO* ontology is the Event Ontology in Figure 4. Therefore, this ontology is not created by this plug-in, it is only populated with instances. The default ontologies are based on the MXML elements illustrated Figure 12. Figure 13 shows the "main skeleton" (or core) of each of 7

default ontologies that are created by this plug-in. The addition of subconcepts and instances for these ontologies happen in the following way:

Figure 11 Screenshot showing the resulting of applying the semantic process mining plug-in Performance Metrics in Ontologies [2] over the results in Figure 10.

1. *DataFieldOntology*: Data fields are part of the MXML element *Data* (cf. Figure 12), and they can be linked to *WorkflowLog*, *Source*, *Process*, *ProcessInstance*, and *AuditTrailEntry*. Therefore, the DataFieldOntology contains a root concept, called "DataField", which has five subconcepts: "LogData", "SourceData", "ProcessData", "ProcessInstanceData", and "AuditTrailEntryData" (to respectively capture the data fields that are linked to the *WorkflowLog, Source, Process, ProcessInstance* and *AuditTrailEntry* MXML elements). Given a log, these ontologies will be extended such that: (i) every data attribute becomes a subconcept of one of the five subconcepts of "DataField". The choice is based on the location of the data attribute in the MXML file. For instance, for the log in Figure 14, a concept "upTime" would be created for the data attribute name "upTime". This concept would be added as a subconcept of "AuditTrailEntryData" because "upTime" is a part of an *AuditTrailEntry* element in the MXML file; and (ii) every data attribute value of a given attribute becomes an instance of the concept that refers to this attribute. As an example, the subconcept "upTime" would have an instance called "96". Actually, the excerpt of the SA-MXML log in Figure 15 illustrates this situation.

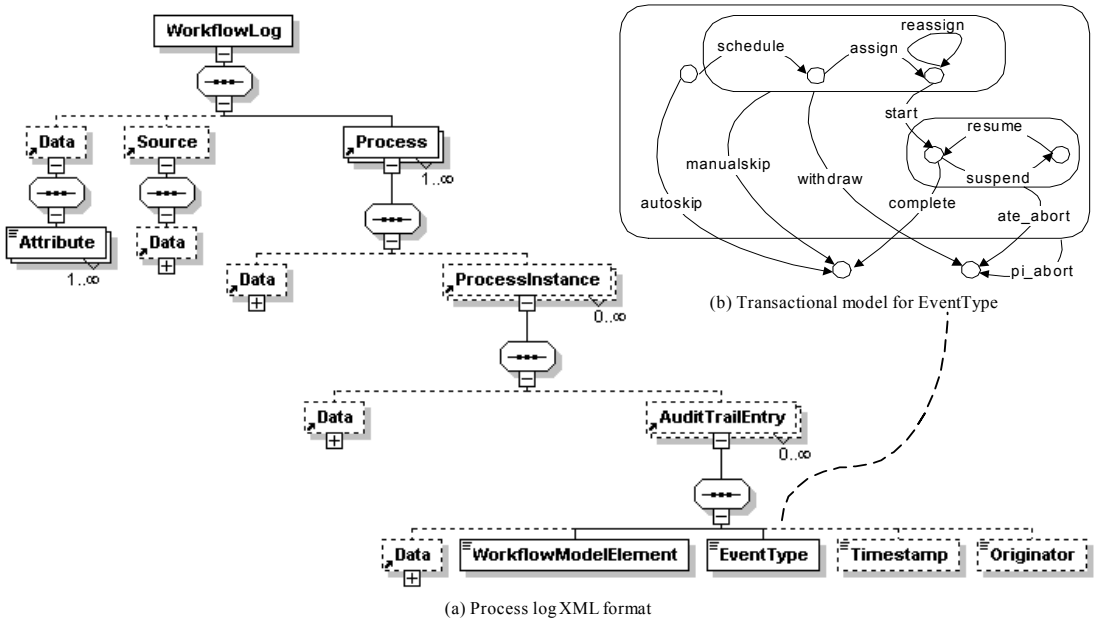


Figure 12 Graphical representation of the Mining XML [13] logging format.

2. *EVO*: Every *EventType* element in a MXML log becomes an instance of a concept in the Events Ontology (EVO) (cf. Figure 4). Note that EVO contains a concept for every event type in the transactional model in Figure 12.

3. *OriginatorOntology*: All *Originator* entries in the event log are mapped to a concept with the same name. All created concepts are a subconcept of the root concept "Originator" (cf. Figure 13). The actual originator names used in the log become instances of these concepts. For instance, for the excerpt in Figure 14, a concept "Mark" would be created. This concept would have one instance, also called "Mark".

4. *ProcessInstanceOntology:* Contains a single "ProcessInstance" concept. All process instances identifiers (i.e., the value of the "id" field) in the event log become an instance of this single concept.

5. *ProcessOntology:* Contains a single "Process" concept, and all processes identifiers in the event become an instance of this single concept.

6. *SouceOntology:* Contains a single "Source" concept, and all values of the field "program"[3] become an instance of this single concept.

7. *TaskOntology:* All *WorkflowModelElement* entries in the event log are mapped to a concept with the name contained in this field. This concept is a subconcept of the root concept "Task". Additionally, every created subconcept has an instance with the same name. For instance, note that the excerpt in Figure 13 contains a *WorkflowModelElement* named "Prepare Gold",

[3] "program" is a required attribute of the element "Source" in the MXML format (cf. http://is.tm.tue.nl/research/processmining/WorkflowLog.xsd).

and the mined *TaskOntology* in Figure 10 indeed contains a subconcept called "PrepareGold" with the instance "Prepare Gold".

8. *WorkflowLogOntology*: Contains a single *WorkflowLog* concept, and a single instance which is the filename of the event log.

Note that both the created default ontologies and the respectively created semantically annotate log (i) are provided within ProM for further analysis by other (semantic) plug-ins, and (ii) can be exported too. The ontologies are exported to WSML files that can be edited with the WSMT tool [17]. The log is exported as a SA-MXML log that links to the mined default ontologies.

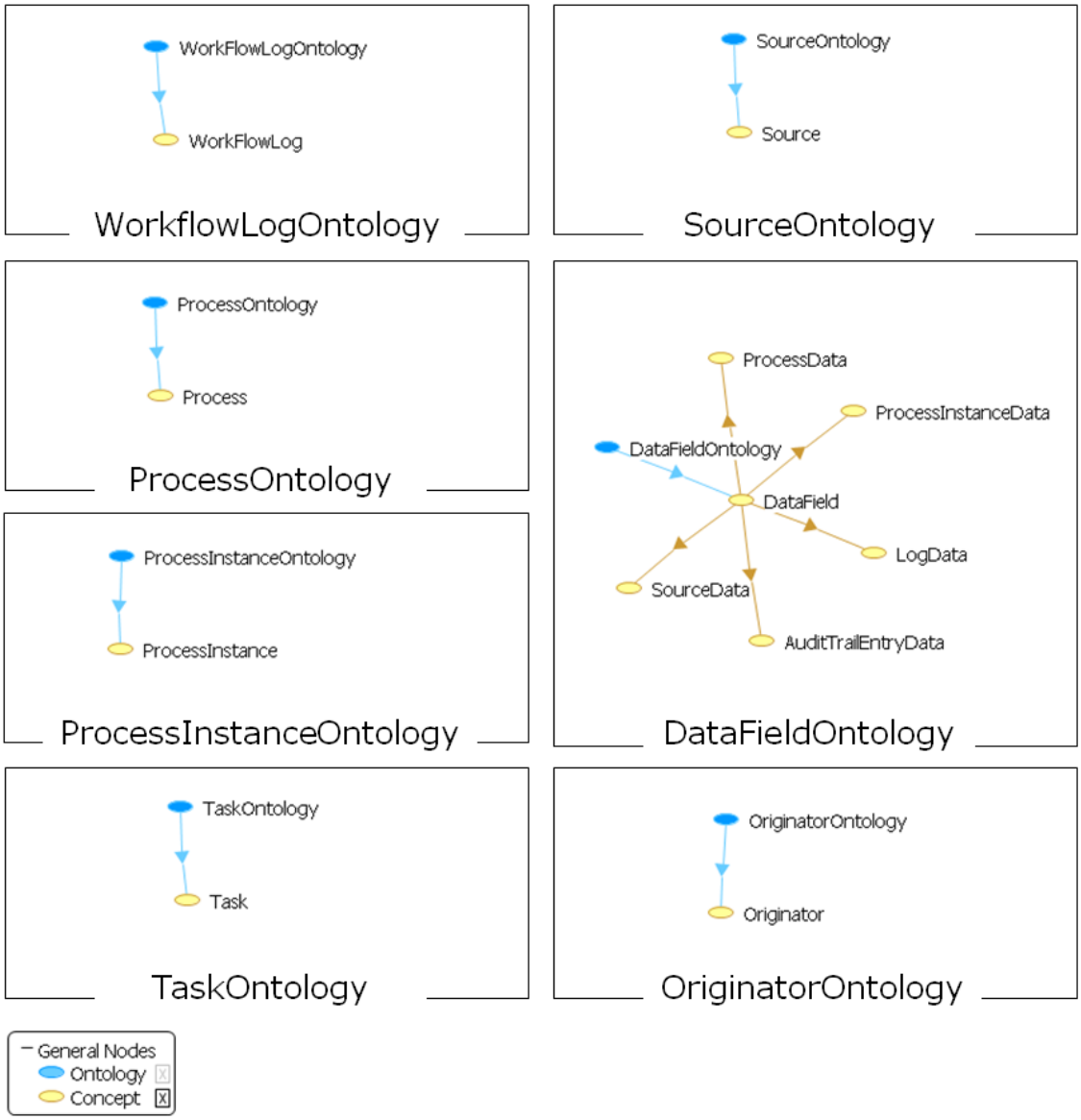


Figure 13 WSMT visualization of the "main skeleton" of 7 default ontologies created by the *Annotate with default ontology* ProM plug-in. The Events Ontology (EVO) is not included become this ontology is defined within SUPER ontology stack (cf. Figure 4).

How to Use the *Annotated with Default Ontologies* Plug-in

To use the *Annotated with Default Ontologies*, perform the following steps in the ProM tool:

1. Open a (SA-)MXML log file by clicking on *File->Open Supported File....*
2. Start the *Annotated with Default Ontologies* plug-in via clicking Analysis-*>LogName->Annotate with default ontologies*
3. If you like, you can export the annotated log and the created default ontologies via clicking *Exports->Annotated log->SA-MXML log file*.

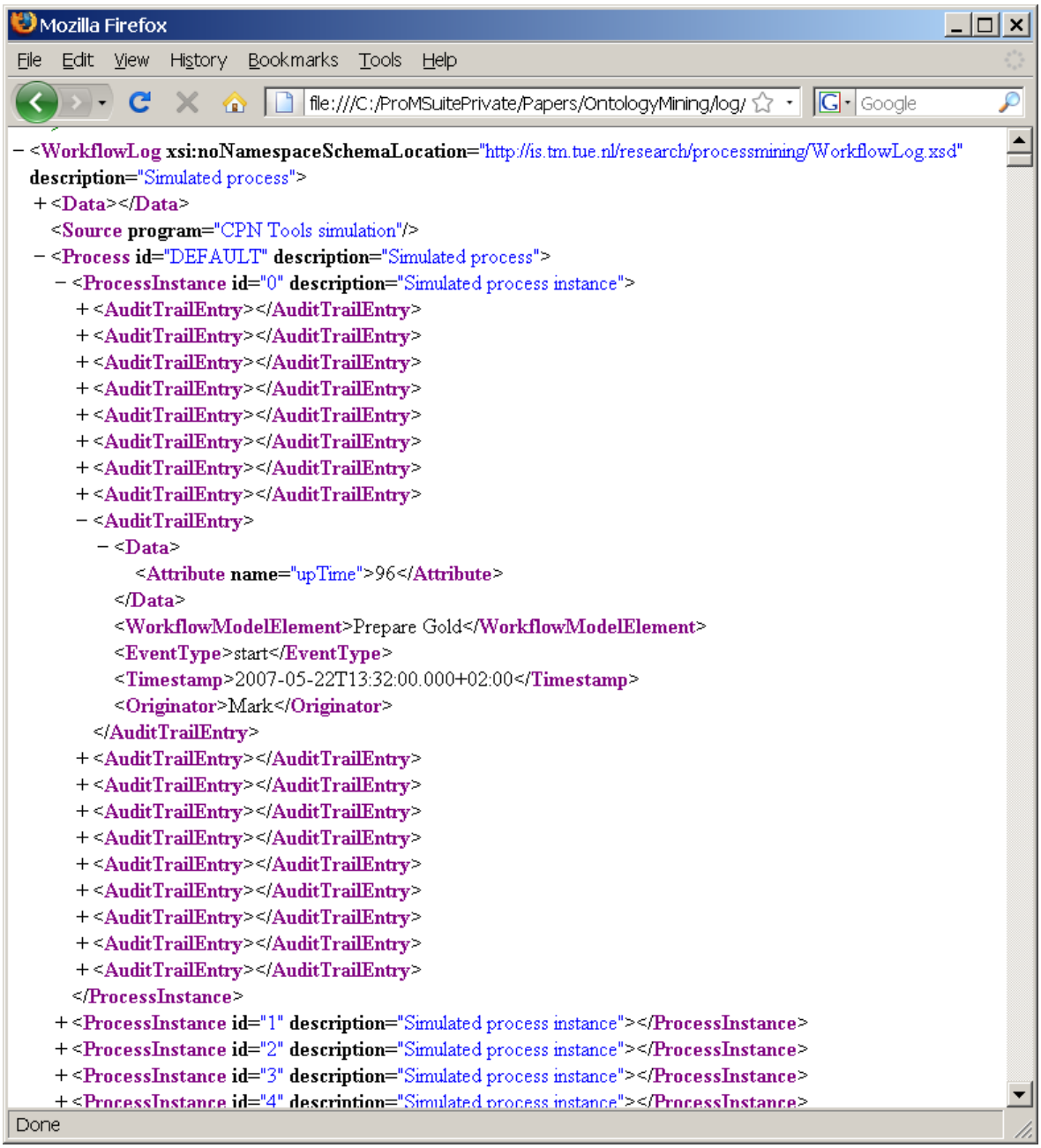
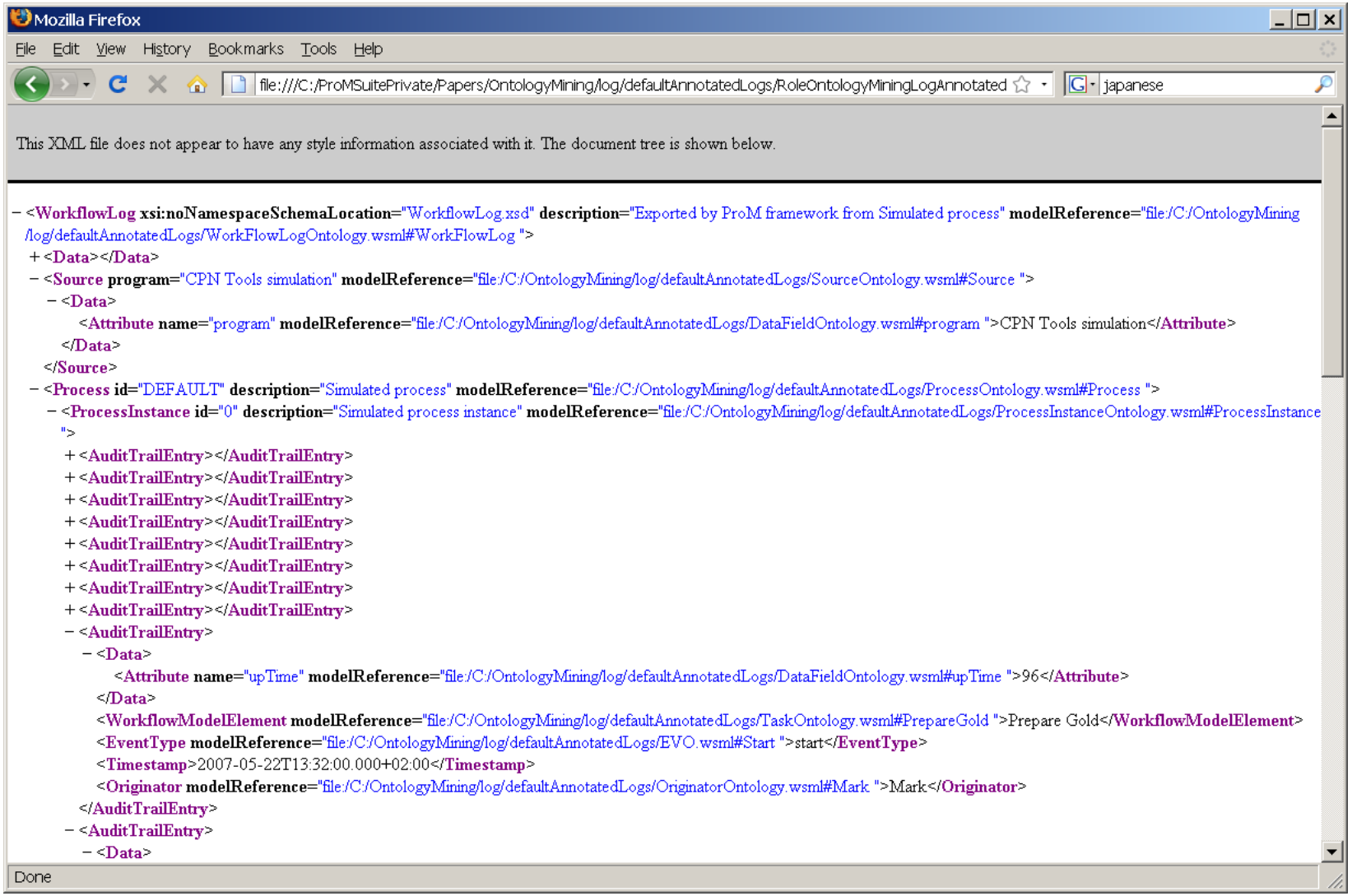


Figure 14 Excerpt of the MXML used to mine the default ontologies in Figure 10.




```
− <WorkflowLog xsi:noNamespaceSchemaLocation="WorkflowLog.xsd" description="Exported by ProM framework from Simulated process" modelReference="file:/C:/OntologyMining
/log/defaultAnnotatedLogs/WorkFlowLogOntology.wsml#WorkFlowLog ">
  + <Data></Data>
  − <Source program="CPN Tools simulation" modelReference="file:/C:/OntologyMining/log/defaultAnnotatedLogs/SourceOntology.wsml#Source ">
    − <Data>
        <Attribute name="program" modelReference="file:/C:/OntologyMining/log/defaultAnnotatedLogs/DataFieldOntology.wsml#program ">CPN Tools simulation</Attribute>
      </Data>
    </Source>
  − <Process id="DEFAULT" description="Simulated process" modelReference="file:/C:/OntologyMining/log/defaultAnnotatedLogs/ProcessOntology.wsml#Process ">
    − <ProcessInstance id="0" description="Simulated process instance" modelReference="file:/C:/OntologyMining/log/defaultAnnotatedLogs/ProcessInstanceOntology.wsml#ProcessInstance
      ">
      + <AuditTrailEntry></AuditTrailEntry>
      + <AuditTrailEntry></AuditTrailEntry>
      + <AuditTrailEntry></AuditTrailEntry>
      + <AuditTrailEntry></AuditTrailEntry>
      + <AuditTrailEntry></AuditTrailEntry>
      + <AuditTrailEntry></AuditTrailEntry>
      + <AuditTrailEntry></AuditTrailEntry>
      + <AuditTrailEntry></AuditTrailEntry>
      − <AuditTrailEntry>
        − <Data>
            <Attribute name="upTime" modelReference="file:/C:/OntologyMining/log/defaultAnnotatedLogs/DataFieldOntology.wsml#upTime ">96</Attribute>
          </Data>
          <WorkflowModelElement modelReference="file:/C:/OntologyMining/log/defaultAnnotatedLogs/TaskOntology.wsml#PrepareGold ">Prepare Gold</WorkflowModelElement>
          <EventType modelReference="file:/C:/OntologyMining/log/defaultAnnotatedLogs/EVO.wsml#Start ">start</EventType>
          <Timestamp>2007-05-22T13:32:00.000+02:00</Timestamp>
          <Originator modelReference="file:/C:/OntologyMining/log/defaultAnnotatedLogs/OriginatorOntology.wsml#Mark ">Mark</Originator>
        </AuditTrailEntry>
      − <AuditTrailEntry>
        − <Data>
```


Figure 15 Excerpt of the SA-MXML that is generated when running the *Annotate with Default Onotologies* plug-in over the log in Figure 14. Note that the elements have been annotated with *modelReference* that link to ontology concepts.

3 Enriching Logs with Semantic Information

3.1 Motivation and Approach

The purpose of semantic annotation is to add formally defined semantics into existing logs so that newly specified semantic process mining tools can use executed over existing logs in companies. Note that the semantic process mining techniques expects log elements to be connected with process ontologies (i.e., to be in the SA-MXML logging format).

Our aim is to automatically recognize the business background ontology instances in logs. Indeed, semantic annotation research is fundamental for semantic Business Process Management Systems (sBPMS). A semantic annotation step is needed to add formal metadata to logs which lack semantic description regarding the related business background. This metadata links data in log to the defined concepts in the business background ontologies. This way, reasoners will be capable of interpreting data in these semantically annotated logs with respect to their respective business background ontologies, as annotated content becomes machine processable. Therefore, our approach proposes an automated annotation using SUPER ontologies for non-(or partially) annotated process logs through *Log Instance Semantics Recognizers (LISR)*.

3.2 Implementation Using ProM

Our semantic annotation ProM plug-in is inspired from Ontos[4], the BYU data-extraction engine, which is a union of many smaller projects, which include: data-extraction ontology development, record boundary detection, table recognition, deep web information extraction, and several others. The process of upgrading the actual process log to be machine-understandable semantic logs is named the process of log annotation. A typical semantic annotation process includes three components. First, a process ontology describes the domain of interest (for instance, the domain ontologies defined for the Nexcom use case). Second, a data instance recognition process discovers all instances of interest in target log files based on the defined ontology. Third, an annotation generation process creates semantic model references to annotate log tags (in our case, these are the workflow model elements, originators, and event types). Note that the added model references provide the ontological context necessary to run the semantic process mining tools.

The following basic presentation shows the capabilities of our plug-in, called *Semantic Log Enricher*. To run this plug-in, users should choose (i) a (partially semantic annotated) log file[5] (see Figure 16), (ii) a business process ontology and (iii) the related Log Instance Semantics Recognizers (see Figure 17), and then ask the tool to do the annotation. During the process, a small window will pop up and show the progress of the annotation process. After the processing is finished, users can view the annotated log file (see Figure 18). The plug-in shows also a report of the annotation process (see Figure 19). This report can be stored in HTML and used for further analysis.

[4] http://www.deg.byu.edu/
[5] Our plug-in can be used to update existing ontology references of already annotated logs. But this functionality is not considered as Log Instance Semantics Recognizer.

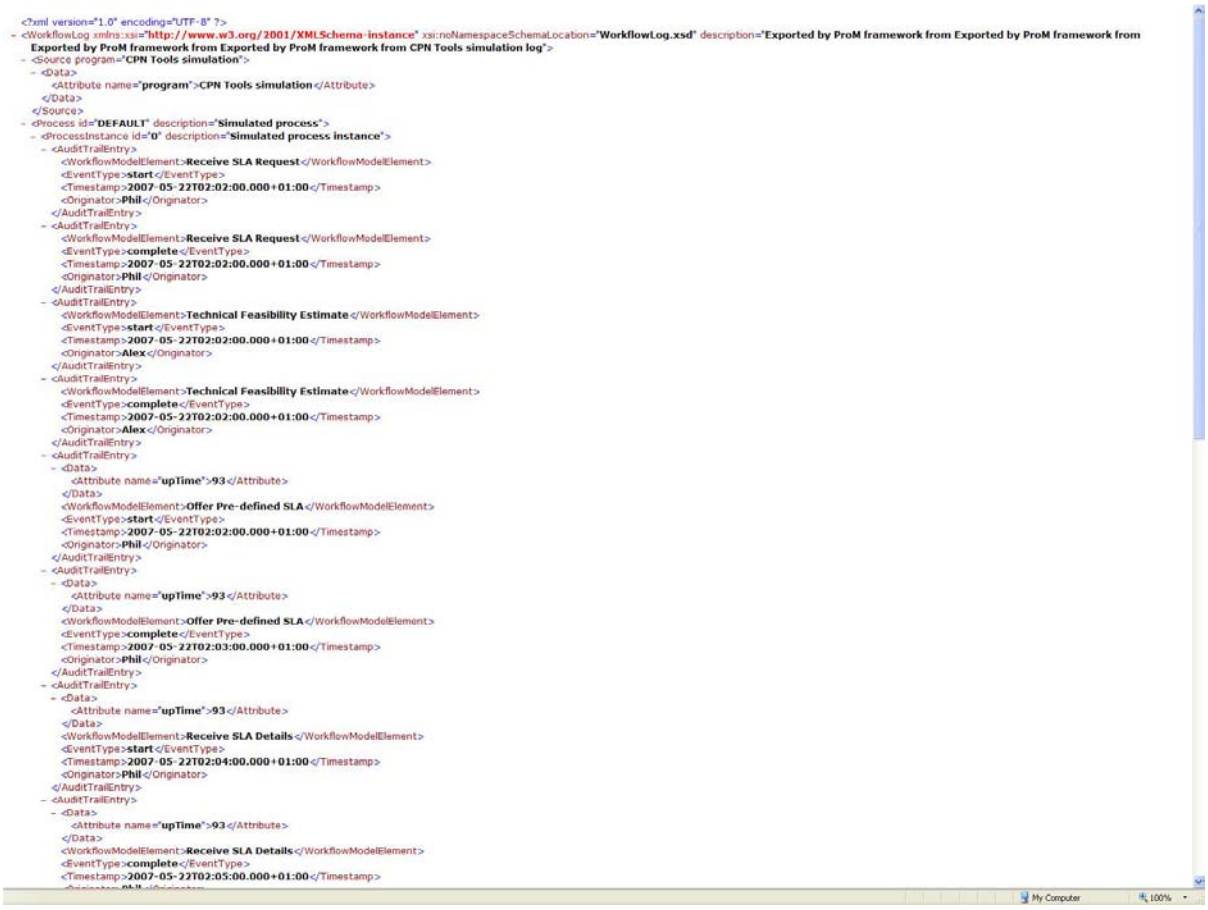


Figure 16 Excerpt of a non-annotated MXML log.

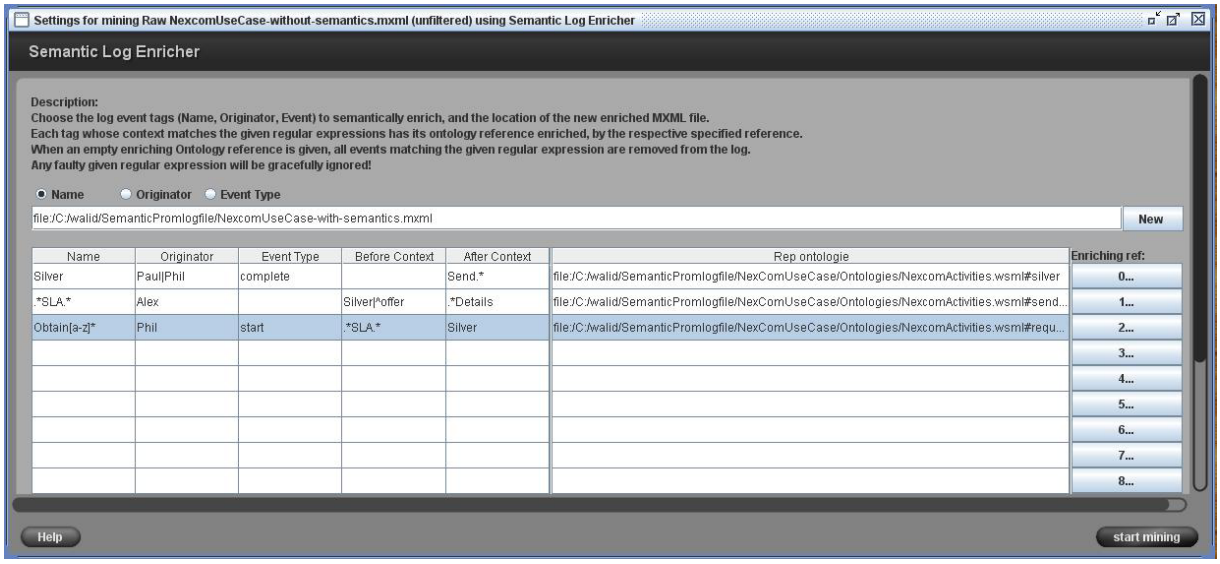


Figure 17 Screenshot showing the main interface of the LISR in the *Semantic Log Enricher* Prom plug-in.

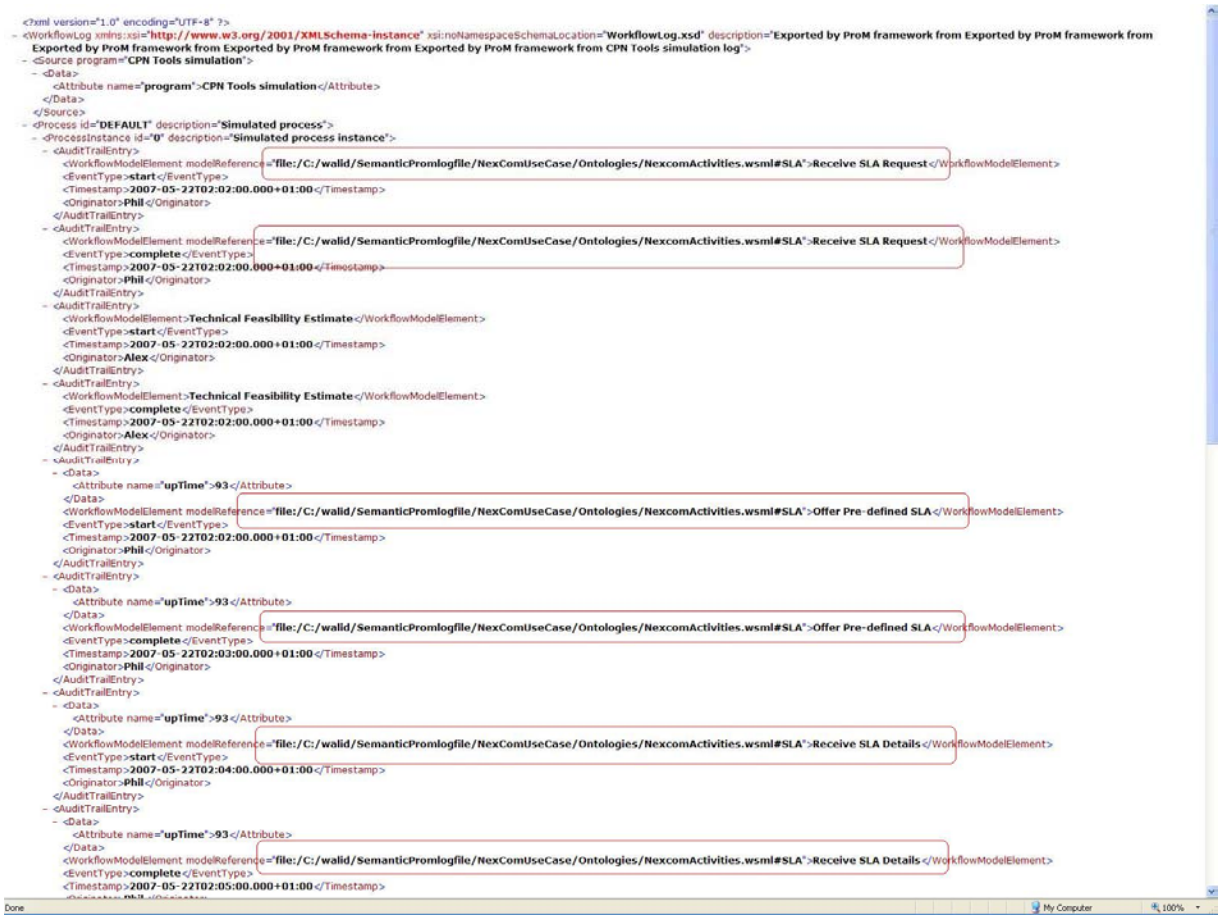


Figure 18 Excerpt of a SA-MXML log. This log has been generated by the *Semantic Log Enricher* plug-in for the log in Figure 16.

LISR are formal specifications that identify instances of a concept in logs. The concept should be a lexical element of a formal ontology (e.g. Nexcom use case concepts such as **`NexcomActivities`**, **`NexcomProcess`**, **`NexcomOrganisationalOntology`**, or more general concepts such as **`Events_Ontology`**, etc.). An LISR of an ontology concept (e.g. **`NexcomActivities.wsml#RequestOffer`**) interprets, for example, the log tag **`<WorkflowModelElement> obtain Availability Quality and Price </WorkflowModelElement>`**, to have the intentional meaning of the defined concept, **`NexcomActivities.wsml#RequestOffer`**. Figure 17 shows the ISR declaration for **`NexcomActivities.wsml#RequestOffer`** concept. The ISR identifies the concept instance using its context (the patterns of its originator, event type, preceding and following log elements). We use Perl-style regular expressions to declare recognition patterns.

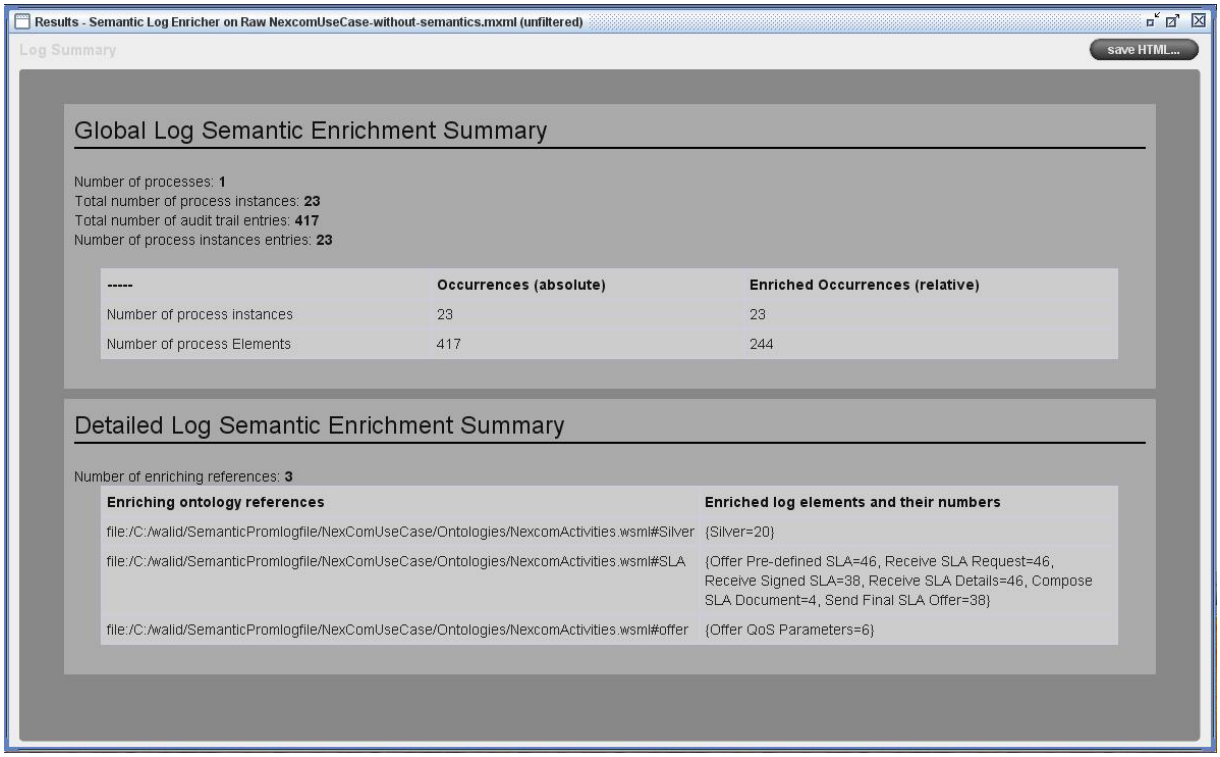


Figure 19 Screenshot showing part of the annotation report in ProM.

In the following, we describe in the form of a table the functionalities provided by the log semantic annotator tool and its dependencies, i.e., the functionalities which this tool expects from the other SUPER components.

Provided Functionalities

Description:	**LISR plug-in semantically annotates MXML log**
Provided to:	Semantic process mining techniques
Input artefacts:	MXML, Ontologies, ISR declaration
Output artefacts:	SA-MXML

ISR Prom Plug-in functionalities

4 Semantic Performance Analysis

Companies usually are interested in knowing how much time is spent in the execution of tasks. This is important because it helps in detecting points of improvements or in building simulations models to analyze different scenarios. In this setting, a critical question is *"How much time is actually spent in the execution of tasks?"*. This question is interesting because many things can happen between the *start* and the *completion* of tasks. Note that human resources do not work 24 hours in a row. Therefore, tasks that start in one day and finish in another have a lot of non-working (or non-processing) hours. Thus, while estimating actual processing times of tasks, techniques that only compute the difference between the completing and starting times are overlooking many characteristics that impact the actual time a task needs to be executed. In this sense, our research within D6.11 has taken information about performers into account when determining the actual processing time of tasks. *Our long-term aim is to come up with more elaborate and robust performance metrics for (semantic) BPM systems*. In the context of this deliverable, this aim is being accomplished in two steps:

1. Define an ontology to support business process analysis. This way, new analysis metrics will be easily incorporated in the SUPER architecture.
2. Define a new semantic performance analysis metric that considers information about tasks performers when calculating execution times.

The COBRA ontology, a Core Ontology for Business pRocess Analysis, is the result of Step 1. This ontology has been created in collaboration with the team responsible for the "Monitoring & Management Tool" of the SUPER architecture (cf. Figure 1). In the Annex we have included the published paper that describes the COBRA ontology. This paper is entitled "A Core Ontology for Business Process Analysis" [19]. The results of performing Step 2 are explained in Subsection 4.1.

4.1 Execution Times Based on Work Shifts

This section explains the new performance metric that we have defined to compute execution times of tasks. The metric is a refinement of the one used to calculate the executions times in the *Performance Analysis with Ontologies* (cf. D5.2 [2], Section 4.6). In this plug-in, execution times are calculated as the time spent between the starting time of a task and its completion time. This means that the availability of performers is not taken into account. Therefore, this section presents new metrics that do take this factor into account.

The remainder of this section is organized as follows. Subsection 4.1.1 describes the defined metric and our approach to calculate this metric. Subsection 4.1.2 introduces the ProM plug-in that implements this metric.

4.1.1 Approach and Metric

Our aim is to calculate the *actual* execution times of tasks. For this, we need to know when performers of tasks have been working on this task. As a result, we have created an approach based on the four steps illustrated in Figure 20:

1. *Calculate Work Shifts*: Companies typically work based on shifts. So, in our approach the end user can define the amount of time included in a shift. Based on this, the work shifts are calculated. For instance, if the work shift time is 6 hours, a day will be divided into four shifts of 6 hours (namely, 0h-5:59h, 6h-11:29h, 12h-17:59h and 18-23:59h). Figure 21 illustrates the idea of shifts in a more abstract way. In this figure, time is divided into work shifts of size x.

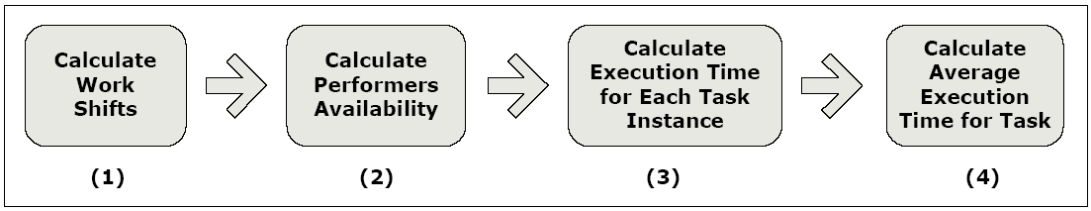


Figure 20 Approach: Four steps to calculate execution times based on performer availability.

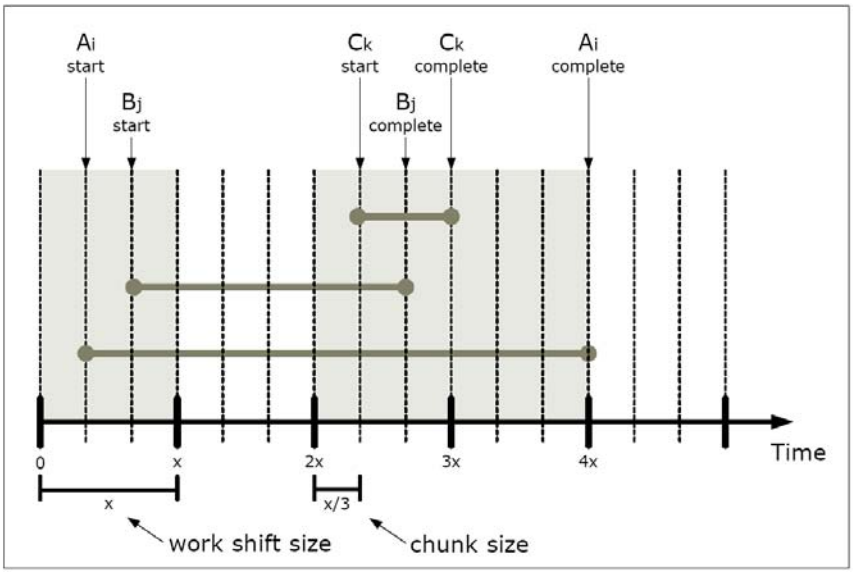


Figure 21 Example of three task instances (Ai, Bj and Ck) to which execution times need to be computed. The instances are performed by a same user. The areas in grey indicate in which work shifts this user was available.

2. *Calculate Performers Availability*: Given that the work shifts are defined, it is necessary to know in which shifts users are available. In our case, we have decided to calculate performers availability based on when events have been registered in the log. So, whenever a performer has a registered event in the log, this perform is considered to be available during this shift. As an illustration, consider the situation in Figure 21, which shows three instances of tasks

that have been executed by a same performer. The grey small circles in the graph indicate when an event has been registered in the log for this user. In this case, 6 events have been registered involving this user. Since these events have happened in the time shifts 0-x, 2x-3x and 3x-4x, we can conclude that this user has been available *only* during these shifts. That is why we have highlighted these shifts in grey in Figure 21. They indicate the user availability.

3. *Calculate Execution Time for Each Task Instance*: Provided that we know when performers are available, we can calculate the actual time that has been spent in executing a certain task instance. We start by dividing work shifts in *equal* chunks of time. The size of the chunk should be such that *"start" events always happen at the beginning of a chunk and "complete" events always happen at the end of a chunk*. We do so because we need to know how many tasks instances a certain performer is executing for a given chunk. Note that users can execute tasks concurrently. Figure 21 illustrates the notion of chunks. In this figure, every chunk has an equal size of x/3. Furthermore, the size of the chunk correctly accommodates the happening of "start" and "complete" events. For instance, we can see that the performer is not execution any task during chunk c1 (from 0 to x/3), is executing task Ai in chunk c2 (from x/3 to 2x/3), is executing two tasks (Ai and Bj) in chunk c3 (from 2x/3 to x), etc.

Given that the notions of user availability, chunk size and the number of tasks in a chunk have been explained, we can now introduce our metric to calculate the actual time spent in the execution of a given process instance. This is given in Definition 3, where the metric *ExecTime* is defined. In a nutshell, the total execution time of a given task instance is calculated by adding up the results of dividing the time of an available chunk (i.e., a chunk in which the user was available) by the number of tasks instances being executed by this user for this chunk. As an example, consider the situation in Figure 21. Note that the total execution time of Ai is equal to (35/18)x (i.e., x*1/3*(1+1/2+1/2+1/3+1/2+1+1+1), which is approximately equal to 1.95x.

Definition 3. (Execution Time Based on User Availability - ExecTime) *Let T be the set of tasks to which all instances have start and complete events registered in a log L. Let T' be the set of task instances in L. Let $\mathcal{T}'$ be the set of all task instances. Let $t' \in T'$ be a task instance of a task $t \in T$. Let C be the set of all chunks for a given period of time including the starting and completion times of all task instances in L. Let $C_{t'} \subseteq C$ be the set of all chunks containing t' and belonging to a work shift in which the performer of t' is available. Let $|c'|$ be the size of a chunk $c' \in C$. Let $\mathcal{C}$ be the set of all chunks. Let $NumberTaskInstancesInChunk : \mathcal{C} \rightarrow \mathbb{N}$ be a function that returns the number of task instances contained in a chunk. Then the execution time based on user availability $ExecTime : \mathcal{T}' \times \mathcal{P}(\mathcal{C}) \rightarrow \mathbb{R}$ metric is defined as follows:*

$$ExecTime(t', C_{t'}) = \sum_{c \in C_{t'}} \frac{|c|}{NumberTaskInstancesInChunk(c)}$$

4. *Calculate Average Execution Time for Task*: The execution time of task is the average of the sum of the execution times of its instances.

So far, we have talked about task instances. However, when concepts are involved, we take into account the instances of concepts. For example, if the task instances in Figure 21 would link to a same concept Y, the average time spent on executing instances of Y would be the *average* of the execution time of Ai, the execution time of Bj, and the execution time of Ck. The next subsection introduces the ProM plug-in that implements our approach for both SA-MXML and MXML logs.

4.1.2 Implementation Using ProM

The analysis plug-in *Execution Times Using Availabiliby Based on Hours Per Shift* implements our approach to calculate execution times based on work shifts (cf. Subsection 4.1.1). In the following, we first describe the functionalities of this plug-in and, afterwards, we explain how to use this plug-in. All screenshots presented in this subsection result from applying this analysis plug-in to the synthetic log at http://is.tm.tue.nl/research/processmining/super/SUPER_NexcomUseCase.zip.

Description

The plug-in *Execution Times Using Availabiliby Based on Hours Per Shift* has the following **15 functionalities**:

Main Interface

The main interface is shown in Figure 22. In this interface, the end user can:

1. Set the number of hours per shift (cf. parameter "Hours per Shift" in Figure 22).
2. Set if ontologies should be considered when calculating the execution times. In this sense, three options are provided (cf. drop-down menu in Figure 22):
    a. *Don't use ontologies*: In this case, the approach described in Subsection 4.1.1 is applied based on the labels for the tasks and originators in the log. The visualization of the results is also provided on graphs built based on these labels (cf. Figure 23).
    b. *Use task ontology*: In this case, the execution times are calculated based on the concepts used to semantically annotate task labels in a log. Note that, when this option is selected, only task instances that are semantically annotated are taken into account during the calculation. The visualization of the results is also provided based on the ontologies that annotate tasks in a log (cf. Figure 24).
    c. *Use originator ontology*: In this case, the calculation of execution times only applies for tasks whose originators (or performers) contain semantic annotations. The results show how long users linked to certain concepts take to execute certain tasks (cf. Figure 25).

3. Calculate the actual execution times based on work shifts (cf. button "Calculate").

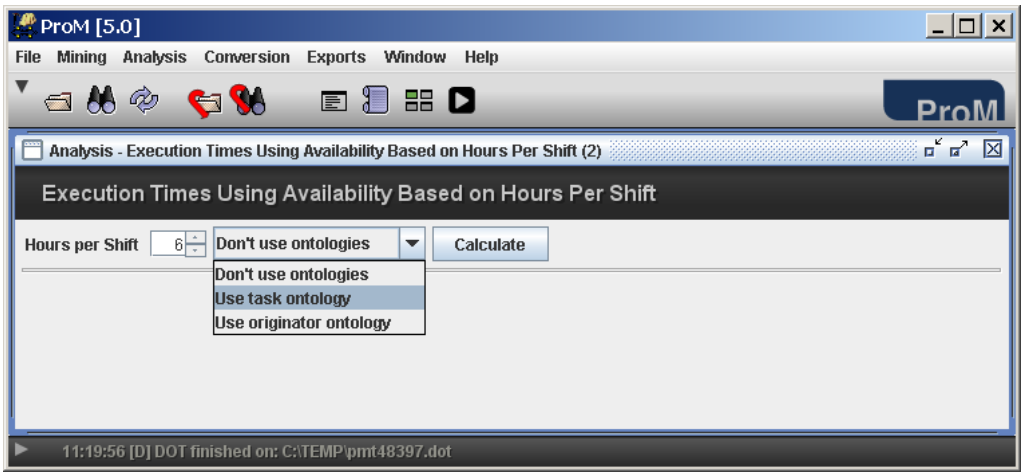


Figure 22 Scheenshot with the main screen of the analysis plug-in *Execution Times Using Availability Based on Hours Per Shift*.

Results Interface

The results interface contains four main tabs: *Graphical View of Execution Times* (cf. Figure 23, 24 and 25), *Originator vs Task* (cf. Figure 26), *Task Statistics* (cf. Figure 27) and *Originator Availability* (cf. Figure 28). The following describes the functionality provided in each tab.

Tab *Originator Availability*

4. Visualize the work shifts in which every performer in the log has been available (cf. Figure 28). A user is available in a shift whenever a *true* value is set in a cell. Note that this table shows the results of performing Step 1 of our approach (cf. Figure 20).
5. Export the *Originator Availability* table to the CSV (Excel) format. This way, end users can build different graphs based on these values.

Tab *Originator vs Task*

6. Visualize how every user is performing while executing instances of tasks. The drop-down menu (cf. Figure 26), provides the following options:
    a. *Average*: Shows the average times that each user takes to execute each task;
    b. *Sum*: Shows the total time a users have spent while executing instances of a given task;
    c. *Frequency*: Shows how often instances of a given task have been executed by a given originator;
    d. *Standard deviation*: Standard deviation for the execution times values;
    e. *Variance*: Variance for the execution times values;
    f. *Minimum*: Minimum time that took a given user to execute an instance of a given task;

g. *Maximum*: Maximum time that took a given user to execute an instance of a given task.

Note that this table shows the aggregated results of performing Step 3 of our approach (cf. Figure 20).

7. Export the *Originator vs Task* table to the CSV (Excel) format. The export is based on the selected view in the drop-down menu.

Tab *Task Statistics*

8. Visualize (in seconds) the execution times values of tasks. This table is an aggregated view of the values in the table *Originator vs Task*. Actually, this table shows the results of performing Step 4 of our approach (cf. Figure 20).

9. Export the *Task Statistics* table to the CSV (Excel) format.

Tab *Graphical View of Execution Times*

10. Visualization of the execution times in a graphical view (cf. Figure 23, 24 and 25).

11. Filtering of which instances to take into account when showing this graphical view (cf. left pane "Filtering" in Figure 23).

12. Visualization of the filtered metric numbers in a tabular way. This functionality is provided by the top-right pane "Numbers" (cf. Figure 23). The colors in the rows of the tables are the same ones used for the graph/ontology in the middle pane. The table has four columns with the following meanings:

a. *Name*: name of the concept/task label in the visualized ontology/graph.

b. *Total*: performance metric as an absolute value.

c. *Frequency*: number of included measurements.

d. *Average*: *Total* divided by *Frequency*.

13. Selection of different units for the visualization of the execution times. This functionality is provided at the bottom-right pane where the option "Number Format" is available.

14. Selection of which column of the table "Numbers" to use when coloring the ontology/graph in the middle pane and table rows. This functionality is provided at the bottom-right pane where the option "Colors based on" is available.

15. Export the table "Numbers" to the CSV (Excel) format.

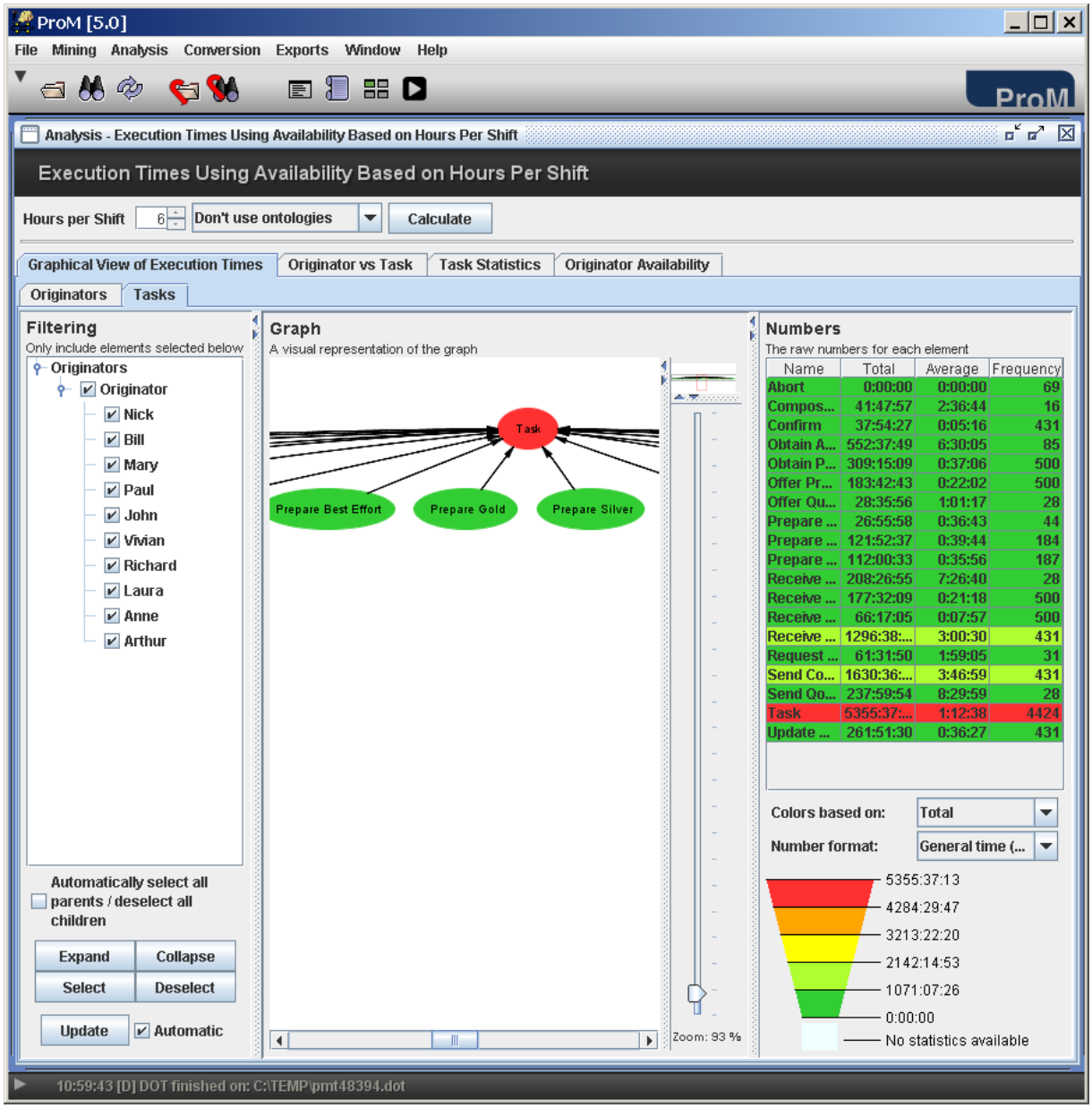


Figure 23 Screenshot showing the tab *Graphical View of Execution Times* when ontologies have not been taken into account. Note that the resulting graphs link each task label to the root node "Task" (cf. middle pane).

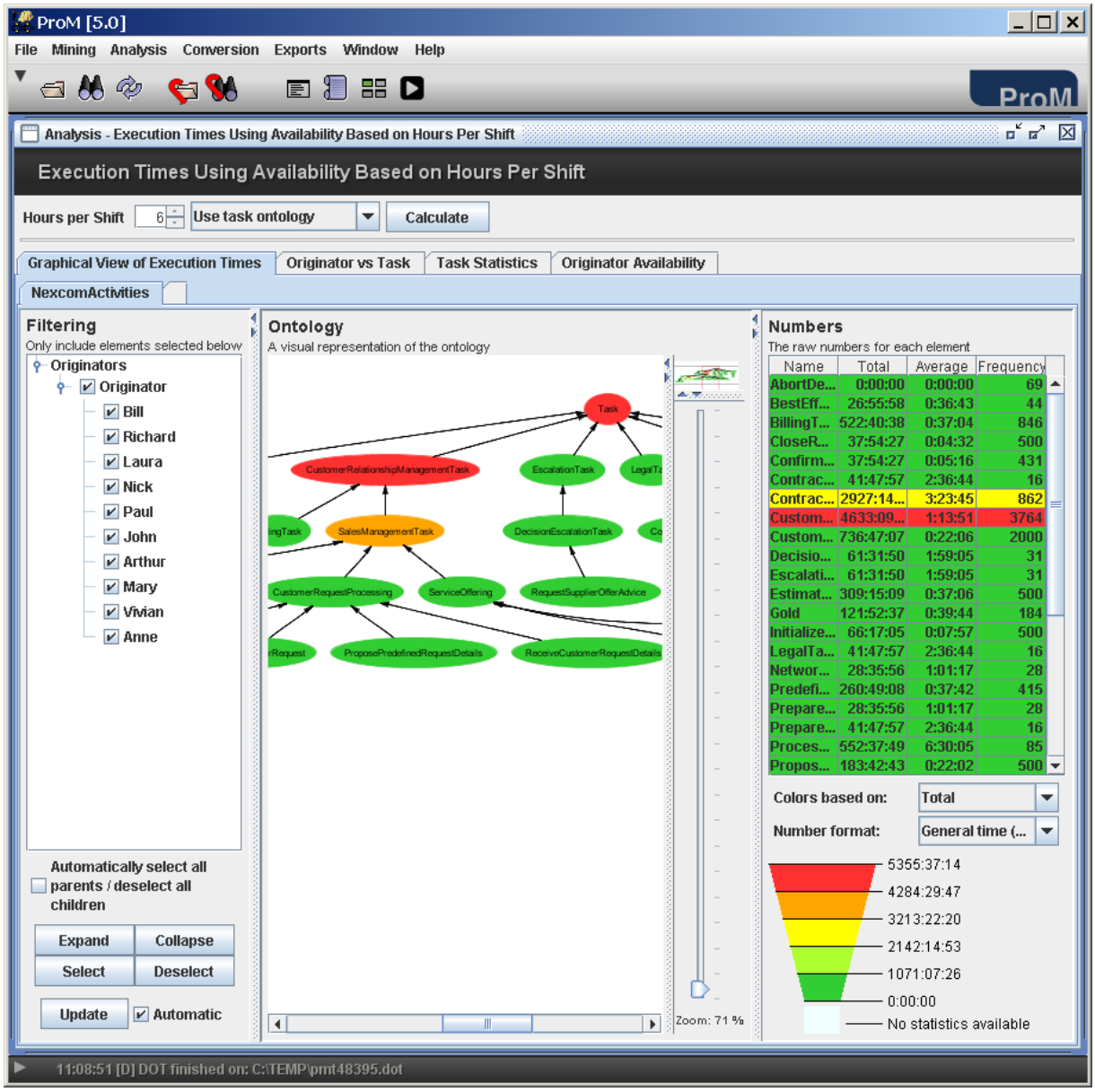


Figure 24 Screenshot showing the tab *Graphical View of Execution Times* based on task ontologies in a log. Note that the task ontology is shown in the middle pane.

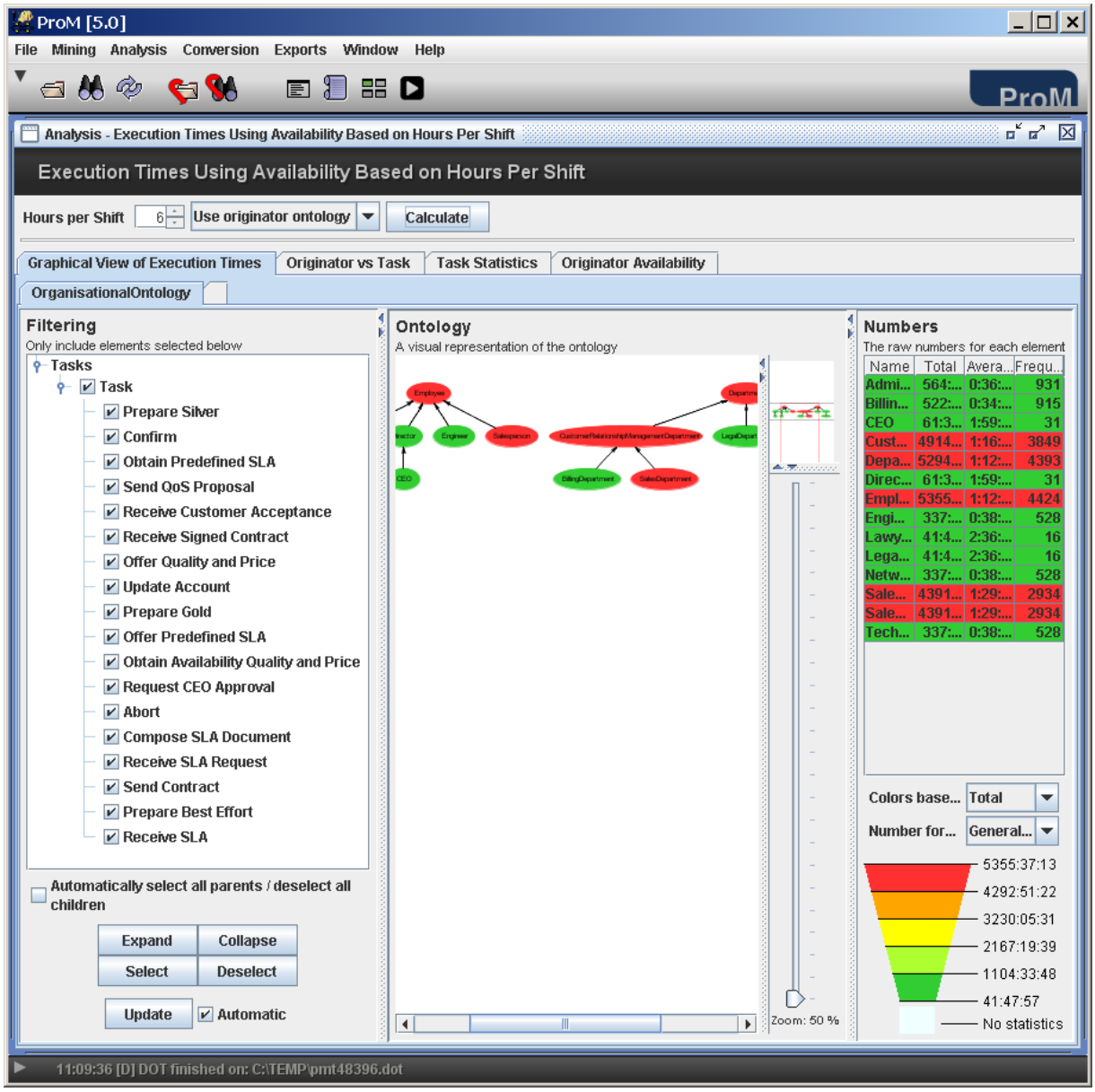


Figure 25 Screenshot showing the tab *Graphical View of Execution Times* when originator ontologies are considered. Note that the originator ontology is shown in the middle pane.

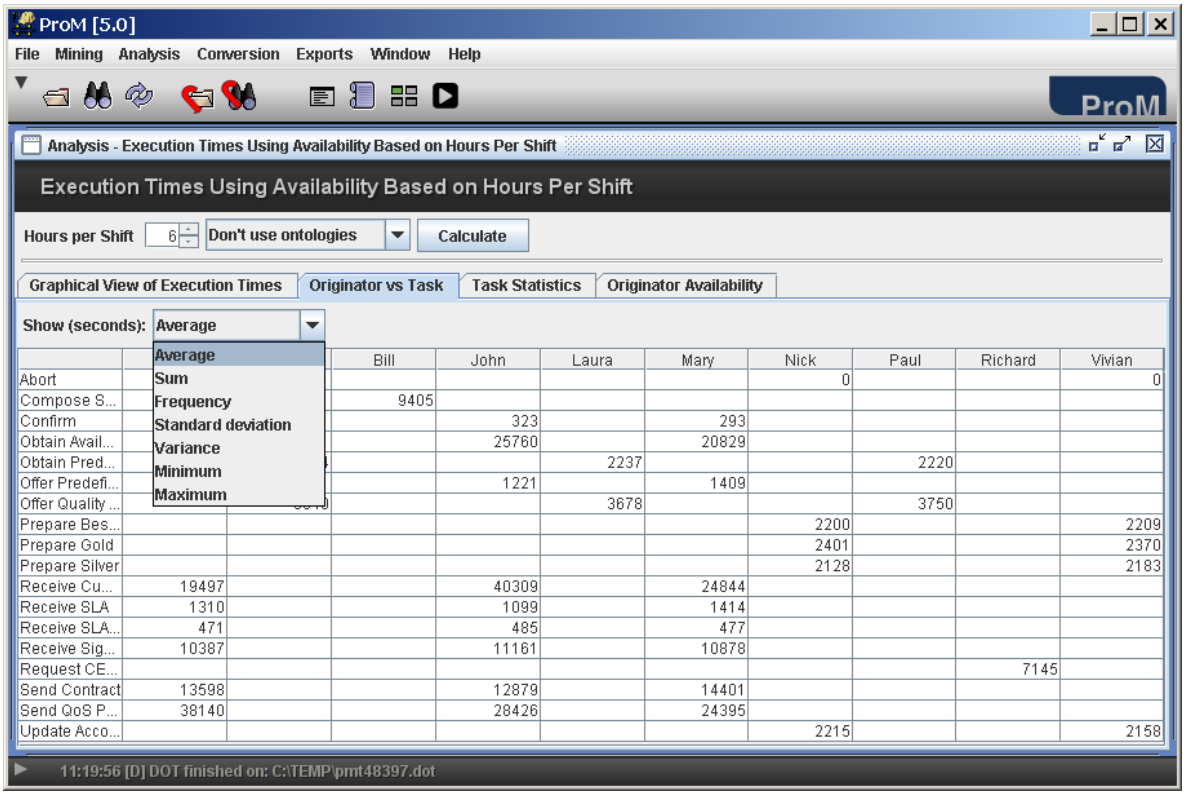


Figure 26 Screenshot showing the results in the tab *Originator vs Task* matrix.

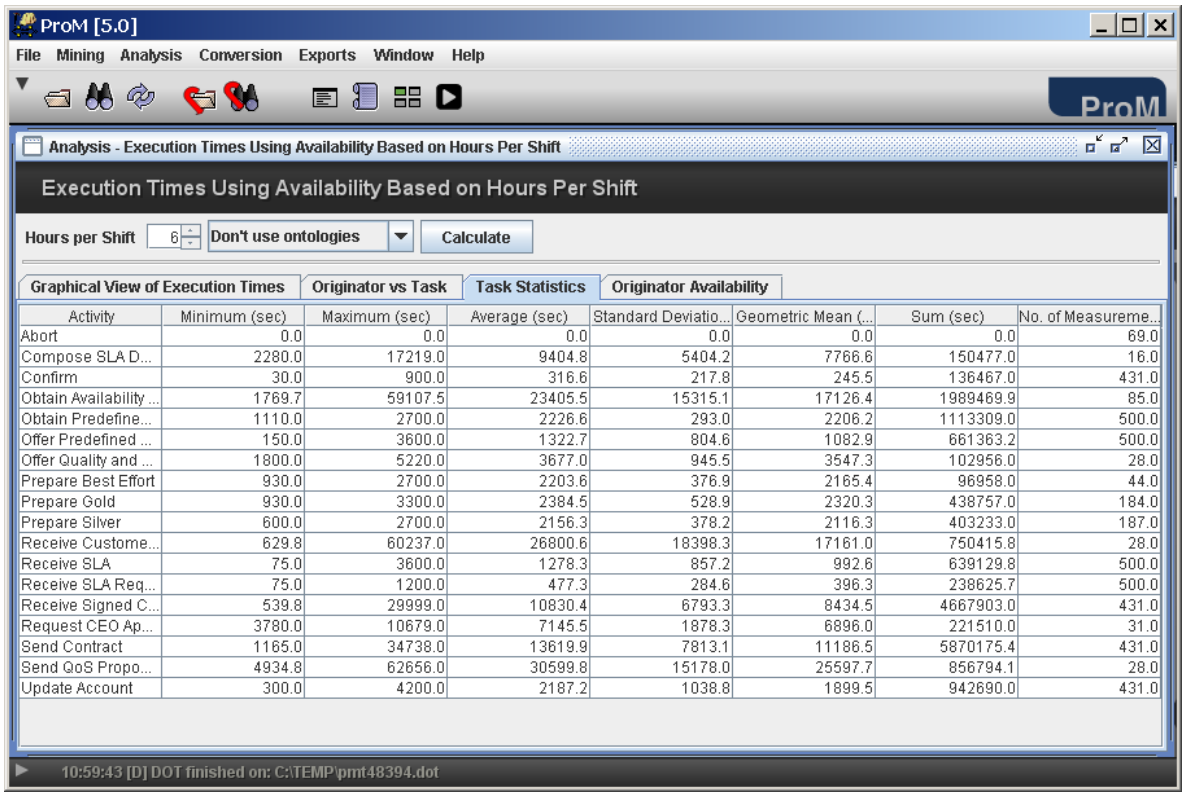


Figure 27 Screenshot showing the results in the tab *Task Statistics*.

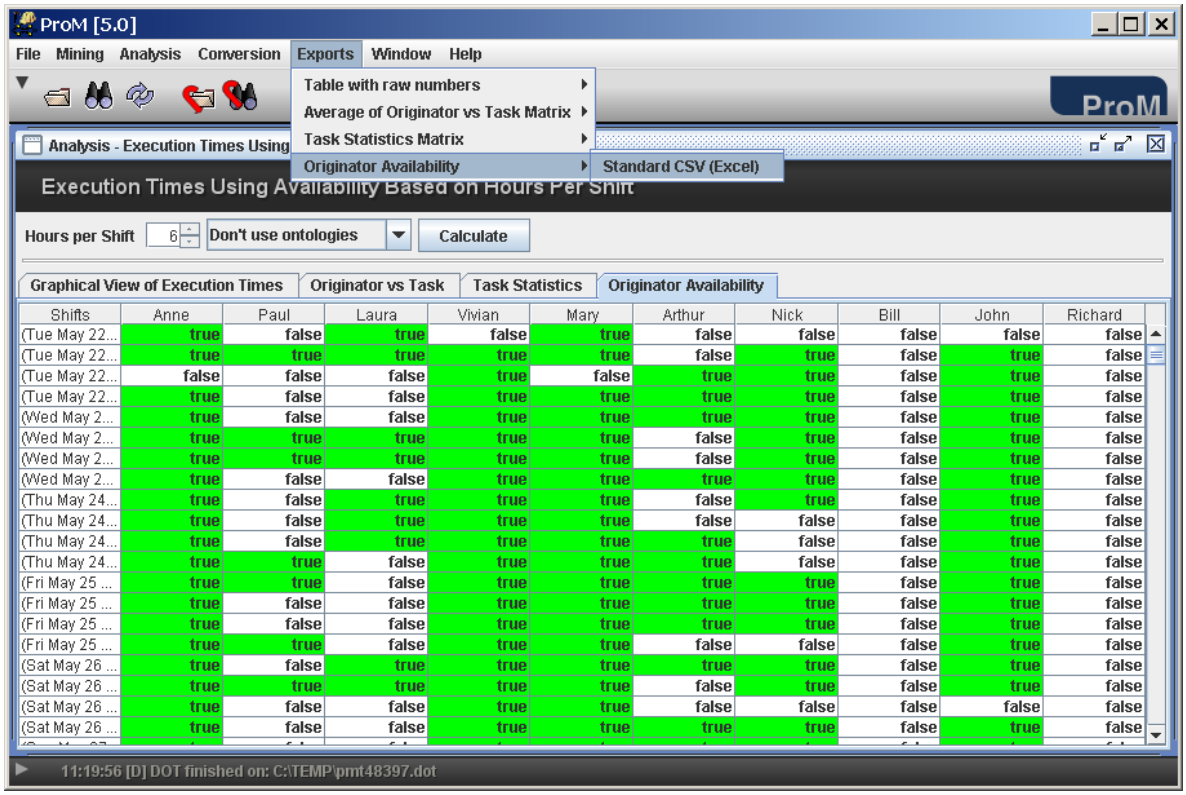


Figure 28 Screenshot showing the results in the tab *Originator Availability*. Note that all the tables shown in four tabs can be exported to the CSV format (cf. menu option "Exports").

How to Use

1. Open a SA-MXML or MXML log file by clicking on *File->Open Supported File...*.
2. Start the *Execution Times Using Availability Based on Hours Per Shift* plug-in via clicking *Analysis-> LogName-> Execution Times Using Availability Based on Hours Per Shift*. You should get a screen like the one in Figure 22.
3. If you like, you can export the table (cf. menu in Figure 28):
    a. "Numbers" (cf. Figure 23) with the (filtered) execution times to CSV format by clicking the menu option *Exports->Table with raw numbers->Standard CSV (Excel)*.
    b. *Originator vs Task* (cf. Figure 26) with the selected view of execution times to CSV format by clicking the menu option *Exports-> <<selected view>> of Originator vs Task Matrix->Standard CSV (Excel)*.
    c. *Task Statistics* (cf. Figure 27) with the execution times to CSV format by clicking the menu option *Exports-> Task Statistics Matrix->Standard CSV (Excel)*.
    d. *Originator Availability* (cf. Figure 28) with the execution times to CSV format by clicking the menu option *Exports-> Originator Availability->Standard CSV (Excel)*.

5 Converter from Execution History to SA-MXML

The converter of the data in the Execution History to the SA-MXML format supported by the ProM tool is implemented as a ProM*import* plug-in. ProM*import* [6] is an open-source tool that facilitates the conversion between the several (commercial) formats used in information systems and the MXML input format of the ProM tool. Within SUPER, we have extended the ProM tool to also have as input event logs in the SA-MXML format [2]. Therefore, to build the converter from the execution history to the SA-MXML format, we have performed the following two steps:

1. Extend the ProM*import* framework to support the export to SA-MXML format.
2. Implement the actual ProM*import* plug-in to convert the data from the Execution History to the SA-MXML format.

As a result, the *SUPER Project* ProM*import* plug-in has been implemented. The main interface of this plug-in is shown in Figure 29. Basically, it contains three input parameters: (i) *Input Directory*, where users can define the directory that contains the Execution History files to be converted. Note that these files are created by the SUPER engine when executing processes. Each file contains WSML instances. When performing the conversion, the ProM*import* plug-in loads these files into the SBP Reasoner (cf. Figure 4) in order to retrieve the WSML instances; (ii) *Input Files Suffix*, where users can define the termination of Execution History files, and (iii) *SA-MXML Output File*, where users can set the name of the resulting SA-MXML to be generated during the conversion. The actual conversion takes place by clicking on the button "Start" (cf. Figure 30). The *SUPER Project* ProM*import* plug-in can be downloaded at http://promimport.sourceforge.net/.

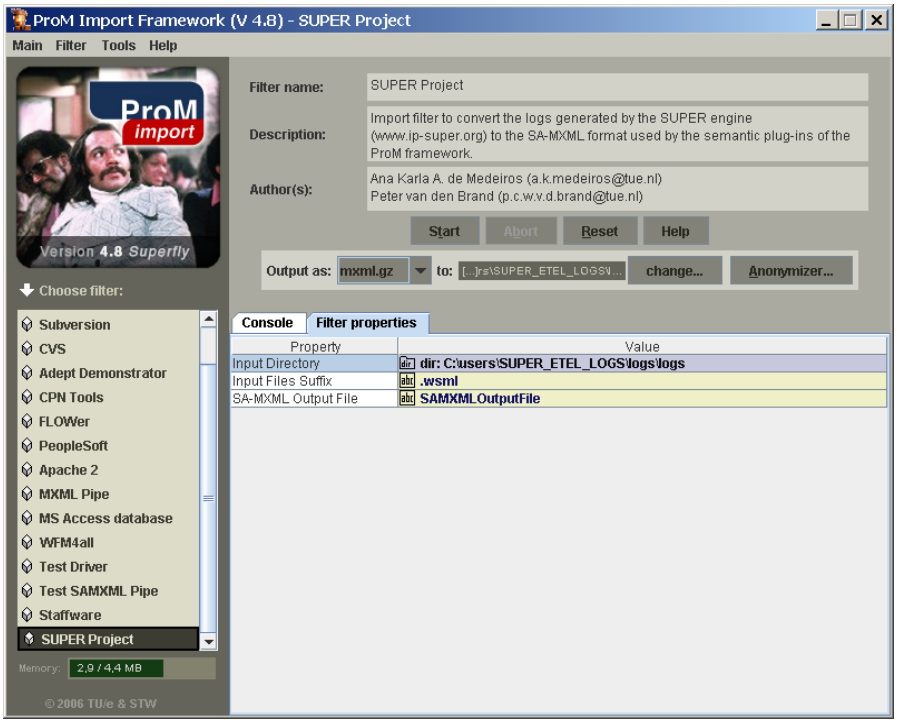


Figure 29 Screenshot of the *SUPER Project ProMimport* plug-in that has been developed within SUPER.

This plug-in converts the data in the Execution History to the SA-MXML used by the semantic process mining tools.

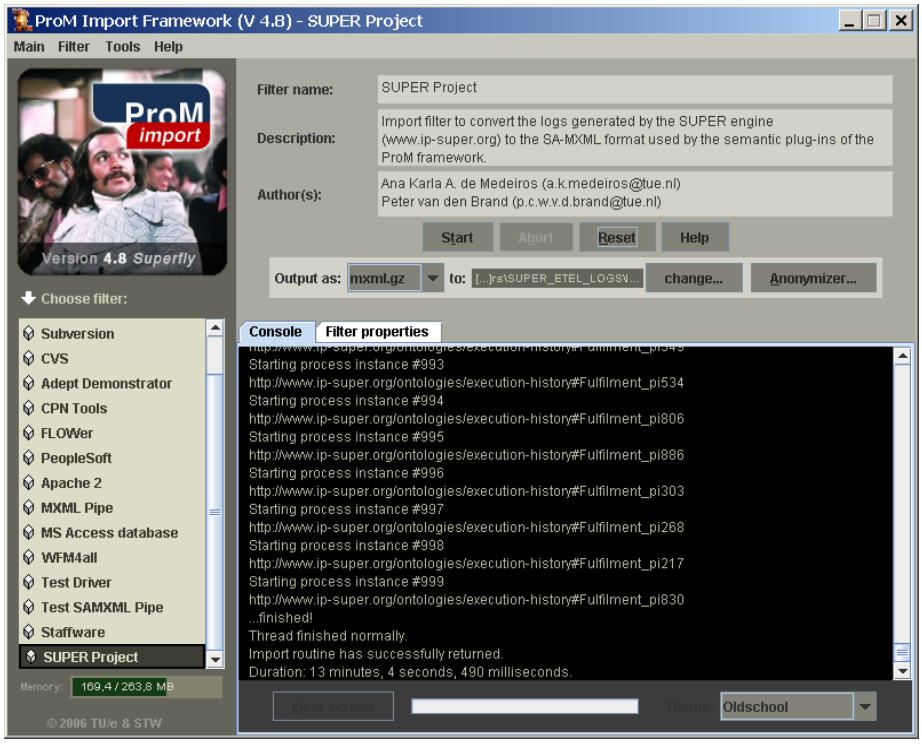


Figure 30 Screenshot showing typical messages that are printed in the "Console" when the *SUPER Project* ProM*import* plug-in is executed.

6 Conclusions

This has described the final implementations we have made for the Semantic Process Mining Tool in SUPER. Together, D5.2, D6.5 and D6.11 explain all the semantic process mining techniques that have been developed in SUPER. In total, *eleven semantic plug-ins* have been developed for the SUPER Process Mining Tools. These plug-ins focus into two aspects: (i) the *actual provision of semantic process analysis* (these are the eight plug-ins: *Ontology Summary*, *Semantic LTL Checker*, *Semantic Control-Flow Mining*, *Semantic Performance Analysis*, *Semantic Organizational Mining*, *Semantic Originator by Tasks Matrix*, *Semantic Ontology URI Renaming Filter* and *Execution Times Using Availability Based on Hours Per Shift*), and (ii) the migration from non-semantic BPM environments to semantic ones (these are the three plug-ins: *Role Hierarchy Miner*, *Annotate with Default Ontologies*, *Semantic Log Enricher*). Additionally, a component has been implemented to convert the History Logs in SUPER to the SA-MXML format used by the semantic plug-ins. This component is the *SUPER Project ProMimport plug-in*. All these plug-ins are freely available together with the ProM and the ProM*import* tool at www.processmining.org.

References

[1] SUPER D5.2 – Semantic Process Instance Environment (http://www.ip-super.org/res/Deliverables/M18/D5.2.pdf).

[2] SUPER D6.5 – Semantic Process Mining Prototype (http://www.ip-super.org/res/Deliverables/M18/D6.5.pdf).

[3] W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems 32(1), 713-732.*

[4] A.K. Alves de Medeiros, W.M.P. van der Aalst, and C. Pedrinaci. Semantic Process Mining Tools: Core Building Blocks. 16th European Conference on Information Systems (ECIS 2008), Galway, Ireland, 2008.

[5] A.K. Alves de Medeiros, C. Pedrinaci, W.M.P. van der Aalst, J. Domingue, M. Song, A. Rozinat, B. Norton, and L. Cabral. An Outlook on Semantic Business Process Mining and Monitoring. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, OTM Workshops (2), volume 4806 of Lecture Notes in Computer Science, pages 1244{1255. Springer, 2007.

[6] C. Günther, W.M.P. van der Aalst. A Generic Import Framework for Process Event Logs. In J. Eder, S. Dustdar, eds.: Business Process Management Workshops. Volume 4103:81-92, 2006.

[7] N. Guarino. Formal ontology, conceptual analysis and knowledge representation. Int. J. Hum.-Comput. Stud. 43(5-6): 625-640 (1995)

[8] A. Lally, and D. Ferrucci. Building an Example Application with the Unstructured Information Management Architecture, 2004, IBM Systems Journal 43, No. 3, 455-475.

[9] SUPER D1.1 – Process Modelling Ontology and Mapping to WSMO (http://www.ip-super.org/res/Deliverables/M12/D1.1.pdf).

[10] SUPER D1.9 – Events Ontology and Core Ontology for Business Process Analysis (http://www.ip-super.org/content/view/32/66/).

[11] The Process Mining (ProM) Framework (http://prom.sourceforge.net/).

[12] H.A. Reijers and S. Limam Mansar. Best Practices in Business Process Redesign: An Overview and Qualitative Evaluation of Successful Redesign Heuristics. Omega: The International Journal of Management Science, 33(4):283-306, 2005.

[13] B.F. van Dongen and W.M.P. van der Aalst. A Meta Model for Process Mining Data. In Proceedings of the CAiSE'05 WORKSHOPS, volume 2. FEUP, 2005.

[14] SUPER D7.2 – Semantic Web Services-based Business Process Architecture. (http://www.ip-super.org/res/Deliverables/M18/D7.2.pdf).

[15] SUPER D9.1 – Requirements Specification for Telecoms Base Environment. (http://www.ip-super.org/content/view/32/66/).

[16] J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The Web Service Modeling Language WSML: An Overview. In Y. Sure and J. Domingue, editors, ESWC, volume 4011 of Lecture Notes in Computer Science, pages 590-604. Springer, 2006.

[17] WSMT: Web Service Modelling Toolkit. http://sourceforge.net/projects/wsmt.

[18] SUPER D6.1 – Execution Engine Design and Architecture. (http://www.ip-super.org/res/Deliverables/M12/D6.1.pdf).

[19] C. Pedrinaci, J. Domingue, A. K. Alves de Medeiros: A Core Ontology for Business Process Analysis. 5th European Semantic Web Conference (ESWC), pages 49-64, Tenerife, Spain, 2008. (Paper available at: http://kmi.open.ac.uk/people/carlos/publications/cobra-eswc2008.pdf)

Annex

In this deliverable we have annexed the original of the following paper:

1. **A Core Ontology for Business Process Analysis**. This paper has been published in the proceeding of the 5th European Semantic Web Conference (ESWC 2008) [19].

A Core Ontology for Business Process Analysis

Carlos Pedrinaci[1], John Domingue[1], and Ana Karla Alves de Medeiros[2]

[1] Knowledge Media Institute, The Open University, Milton Keynes, MK7 6AA, UK.
E-mails:{c.pedrinaci, j.b.domingue}@open.ac.uk
[2] Eindhoven University of Technology, P.O. Box 513, 5600MB, Eindhoven,
The Netherlands. E-mail: a.k.medeiros@tue.nl

Abstract. Business Process Management (BPM) aims at supporting the whole life-cycle necessary to deploy and maintain business processes in organisations. An important step of the BPM life-cycle is the analysis of the processes deployed in companies. However, the degree of automation currently achieved cannot support the level of adaptation required by businesses. Initial steps have been performed towards including some sort of automated reasoning within Business Process Analysis (BPA) but this is typically limited to using taxonomies. We present a core ontology aimed at enhancing the state of the art in BPA. The ontology builds upon a Time Ontology and is structured around the process, resource, and object perspectives as typically adopted when analysing business processes. The ontology has been extended and validated by means of an Events Ontology and an Events Analysis Ontology aimed at capturing the audit trails generated by Process-Aware Information Systems and deriving additional knowledge.

1 Introduction

Many companies use information systems to support the execution of their business processes. Examples of such information systems are Enterprise Resource Planning, Customer Relationship Management, and Workflow Management Systems (WFMS). These systems usually generate events while executing business processes [1] and these events are recorded in logs. The competitive world we live in requires companies to adapt their processes in a faster pace. Therefore, continuous and insightful feedback on how business processes are executed becomes essential. Additionally, laws like the Sarbanes-Oxley Act force companies to show their compliance to standards. In short, there is a need for good analysis tools that can provide feedback about how business processes are actually being executed based on the observed (or registered) behaviour in event logs.

BPM results from the limitations exhibited by WFMS which mainly focus on the enactment of processes by generic engines and does not take into account the continuous adaptation and enhancement of existing processes. BPM acknowledges and aims to support the complete life-cycle of business processes which undoubtedly involves post-execution analysis and reengineering of process models. A key aspect for maintaining systems and the processes they support

is the capability to analyse them. BPA is particularly concerned with the behavioural properties of enacted processes may it be at runtime, as in Business Process Monitoring, or post-execution as in Business Process Mining [1] or Reverse Business Engineering.

Due to its cyclic nature, BPM has however made more evident the existing difficulties for obtaining automated solutions from high-level business models, and for analysing the execution of processes from both a technical and a business perspective. The fundamental problem is that moving between the business-level and the IT-level is hardly automated [2]. Deriving an IT implementation from a business model is particularly challenging and requires an important and ephemeral human effort which is expensive and prone to errors. Conversely analysing automated processes from a business perspective, e.g., calculating the economical impact of a process or determining the performance of different departments in an organisation, is again an expensive and difficult procedure which typically requires a human in the loop. *Semantic Business Process Management* (SBPM), that is the combination of Semantic Web and Semantic Web Services technologies with BPM, has been proposed as a solution [2].

In this paper we present results obtained in the context of the European project SUPER (IST-026850) which aims at developing a SBPM framework, based on Semantic Web Services technology, that acquires, organises, shares and uses the knowledge embedded in business processes in order to make companies more adaptive. This semantic framework will support the four phases of the BPM life-cycle and the research presented in this paper provides the foundation for semantic BPA. *In particular we shall describe a core ontology for business process analysis which bridges the gap between low-level monitoring information and high-level business knowledge.* The remainder of the paper is organised as follows. Section 2 reviews existing research that makes use of semantic technologies and present a set of requirements and competency questions that semantic BPA technologies should address. Section 3 presents COBRA, a Core Ontology for Business pRocess Analysis, and Time Ontology which provides the basis for using temporal reasoning within BPA. Section 4 illustrates how COBRA can be applied to BPA. Section 5 presents our conclusions and describes future research to be carried out.

2 Semantics in Business Process Management

In the last years significant efforts have been devoted to integrating automated reasoning with the BPM domain, a field where the application of knowledge-based technologies appears to be the next evolutionary step [3]. These efforts can roughly be divided into top-down and bottom-up approaches. Top-down approaches make use of high-level conceptual models to structure and reason about Business Process Management activities. Among these approaches we find research on enterprise ontologies, models for resources consumption and provision, value flows, service bundling, etc. [2, 4–8]. However, despite the variety of models and tools produced so far there is little uptake within the industry which

is often due to the existing difficulty to provide and maintain good knowledge bases expressed in terms of these conceptual models.

On the other hand, bottom-up approaches integrate some sort of light-weight automated reasoning machinery with existing BPM solutions, see for instance [9–11]. These efforts are mainly dominated by researchers from the BPM area, where knowledge-based technologies have not been widely used so far. The focus has mainly been the annotation of data warehouses or the application of rule engines to control resources and ensure certain business policies are followed. Unfortunately, the information manipulated is mostly in syntactic formats which is hardly amenable to automated reasoning. In fact, most of the budget when applying so-called Business Intelligence solutions is typically devoted to the manual integration of data from BPM systems and this is often an ephemeral effort which has to be repeated over time. As a result the benefits gained by applying these techniques are largely limited. Still, as opposed to top-down approaches, the fact that these research efforts are grounded into deployed BPM systems increases their impact in the industry.

What can be distilled from the current state-of-the-art is that the existent epistemological gap between, on the one hand industry BPM solutions, and on the other hand knowledge-based research, hampers to an important extent the wider application of semantics in BPM. The research presented in this paper aims precisely at reducing this gap when it comes to analysing business process executions. In order to guide and validate our approach we present next a representative set of requirements and competency questions that we have identified based on existing practice within the BPM domain.

2.1 Requirements for Semantic Business Process Analysis

BPA is typically structured around three different views: (i) the *process view*; (ii) the *resource view*; and (iii) the *object view* [12]. The process view is concerned with the enactment of processes and is thus mainly focussed on the compliance of executed processes with respect to prescribed behaviours and Key Performance Indicators that can support business analysts in the examination and eventual optimisation of deployed processes [1]. Relevant information in this respect are (i) "the processes and activities currently running"; (ii) "which ones have been completed and whether they were successful or not"; (iii) "the execution time of the different business activities"; (iv) "which business activities have preceded which others", etc. The resource view is centred around the usage of resources within processes. In this perspective, the performance at different levels of granularity (individuals, organisational units, etc.), work distribution among the resources, and the optimisation of resources usage are the main aspects analysed. Typical questions in this perspective are for instance (i) "which resources were involved in which business activities"; (ii) "which actor was responsible for a certain process"; (iii) "which external providers appear to work more efficiently"; (iv) "what's the average number of orders processed by the sales department per month", etc. Finally, the object view focusses on business objects such as inquiries, orders or claims. This perspective is often adopted in

order to better analyse the life-cycle of so-called Business Objects. In this perspective, business analysts often want answers to questions like (i) "what is the average cost per claim"; (ii) "which is the item we are currently selling the most (or the least)"; (iii) "what's the overall benefit we are obtaining per item"; (iv) "are critical orders processed in less than two hours", etc.

These three views are populated with statistical information such as the minimum, the average or the deviation of some parameter of interest, and correlations are typically established across them, e.g., "what is the average process execution time for processing each type of order?". Common to these scenarios where BPA techniques are applied is the underlying dependency with respect to time manipulation (e.g.,"are critical orders processed in less than two hours"), the need to navigate through different levels of abstraction (e.g., "what's the average number of orders processed by the sales department per month") and across the different perspectives, and the overall necessity to apply general purpose methods over domain specific data.

Therefore, to enhance the state-of-the-art of BPA we need a comprehensive conceptual model of the BPM domain that supports applying general purpose knowledge-based techniques over domain specific data, coupled with the capacity to navigate through different levels of abstraction across the process, resource, and object perspectives, and the ability to appropriately deal with temporal aspects. The next section is devoted to presenting a core ontology for supporting Business Process Analysis that aims to provide a generic and extensible conceptual model that can support the competency questions exposed above.

3 An Ontology for Business Process Analysis

In order to support the level of automation required by enterprises nowadays we need to enhance BPA with support for applying general purpose analysis techniques over specific domains in a way that allows analysts to use their particular terminology and existing knowledge about their domain. To this end we have defined the Core Ontology for Business pRocess Analysis. COBRA provides a core terminology for supporting BPA where analysts can map knowledge about some particular domain of interest in order to carry out their analyses. It is worth noting that COBRA does not aim to provide a fully-comprehensive conceptualisation for supporting each and every kind of analysis since the scope would simply be too big to be tackled appropriately in one ontology. Instead COBRA provides a pluggable framework based on the core conceptualisations required for supporting BPA and defines the appropriate hooks for further extensions in order to cope with the wide-range of aspects involved in analysing business processes. These extensions are currently been developed in SUPER as part of an ontological framework aimed at providing an extensive conceptualisation of the BPM domain ranging from process modelling to the definition of business strategies. Still, COBRA already provides a good basis for supporting the most typical analysis as described in the previous section.

COBRA has been developed using the Operational Conceptual Modelling Language (OCML) [13], which provides support for executing the definitions in

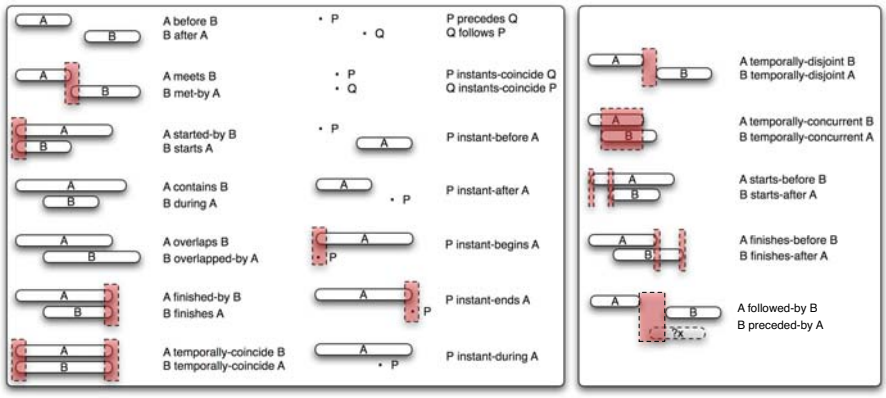


Fig. 1. Instants and Interval relations.

the ontology as well as export mechanisms to other representations including OWL and WSML. COBRA builds upon two ontologies, namely Base Ontology and Time Ontology, and is currently enhanced with Events Ontology for capturing audit trails, and Events Analysis Ontology which provides a set of generic reusable rules and relations[3]. Base Ontology provides the definitions for basic modelling concepts such as tasks, relations, functions, roles, numbers, etc. The interested reader is referred to [13] for further information. The other ontologies will be described in the remainder of this section.

3.1 Time Ontology

COBRA builds upon Time Ontology that provides a temporal reference by means of which one can determine temporal relations between elements. The ontology defines three top-level concepts, namely *Time Instant*, *Time Interval*, and *Temporal Entity*. Time Instant is the main primitive element and it provides the means for identifying a point in time with precision up to the microsecond for we aim to support monitoring automated systems. Time Intervals are defined by means of the start and end instants and have therefore an associated duration which can be computed by means of a function that substracts the limiting instants. Temporal Entity, as opposed to the conceptualisation proposed in [14], represents entities that have a temporal occurrence, and are therefore different from Time Instant and Time Interval which are the base constructs that represent a particular point or period in time.

Using these core concepts we have implemented the interval relations defined by Allen [15], the additional instant-interval relations defined by Vilain [16], and useful functions for computing the duration of intervals or for obtaining the current Time Instant. The left hand-side of Figure 1 illustrates these relations, whereby A and B represent Time Intervals, whereas P and Q represent Time Instants. The relations are self-explanatory, the interested reader is referred to [15] and [16] for further details. It is worth noting that we have renamed the equality

[3] The ontologies can be found at http://kmi.open.ac.uk/people/carlos

relations for Time Intervals and Time Instants to *Temporally Coincide* and *Instants Coincide* respectively, for we believe it is counterintuitive to use the term "*equal*" for referring to different things that occur at the same time.

In addition to these relations we have also included for convenience a few typical disjunctions of Allen's algebra, e.g., *Before-Or-Meets*, and further relations which are relevant for BPA. The latter are depicted in the right-hand side of Figure 1. The new relations we have implemented are namely *Temporally Disjoint*, *Temporally Concurrent*, *Starts-Before*, *Starts-After*, *Finishes-Before*, *Finishes-After*. Two intervals are considered to be Temporally Disjoint if there is no interval shared between the two, which in Allen's interval algebra is equivalent to a disjunction between Before, After, Meets and Met-By. Temporally Concurrent is the inverse relation of Temporally Disjoint and it therefore holds when there exists some interval shared between the two concurrent intervals. Starts-Before, Starts-After, Finishes-Before and Finishes-After, which we believe are self-explanatory, are based on the numerical comparison between the start instant or end instant of the intervals.

Our Time Ontology considers two kinds of *Temporal Entities*, namely *Instantaneous Entity* and *Time Spanning Entity*. Instantaneous Entities are phenomena that occur at a specific point on time and whose duration can be neglected. By contrast, Time Spanning Entities are those that last over a period of time indicated by the *spansInterval* slot. The distinction between, on the one hand, Temporal Entities and, on the other hand, Time Instant and Time Interval allows us to apply the previous relations over a plethora of entities, i.e., every Temporal Entity. In addition to the previously mentioned relations we have included two which are specific to Time Spanning entities and are particularly useful for BPA, namely *Followed-By* and *Preceded-By*. A Time Spanning Entity I is *Followed-By* by another Time Spanning Entity J of kind C, if J is After I and there is no other Time Spanning Entity X of kind C which is After I and Starts-Before J. *Preceded-By* is the inverse relation of *Followed-By*.

One of the main characteristics of our Time Ontology is the use of *polymorphism*. Our ontology supports determining temporal relations about the primitive elements Time Instant and Time Interval, between these and Temporal Entities, and between any two Temporal Entities. To do so, the relations have been implemented on the basis of backward-chaining rules that perform the appropriate transformations between Temporal Entities and their primitive counterpart and then invoke the primitive relation. This polymorphism is a convenient feature of our conceptualisation in order to support BPA where temporal relations need to be evaluated between executions of activities, e.g., "was Activity A executed after Activity B?", executions of processes, e.g.,"has Process A been run concurrently with Process B", but also with respect to reference intervals or instants, e.g., "retrieve all the Processes executed in the last month".

3.2 Core Ontology for Business Process Analysis

We previously introduced that BPA is concerned with the analysis of the execution of business processes from several perspectives. In particular, we identified

the *process view*, the *resource view*, and the *object view*. COBRA has therefore been structured around these very views in an attempt to enhance BPA with support for the automated reasoning, querying, and browsing of audit trails from different perspectives and at different levels of abstraction. The ontology is depicted in Figure 2 using an extended UML notation where arrows represent the *isA* relation, dashed arrows denote the *instanceOf* relation, and lines represent custom relations. Further notation extensions will be explained as the need arises during the description of the ontology.

The development of COBRA has been guided to an important extent by existing ontologies like the Enterprise Ontology [5], DOLCE [17], TOVE [4] and CIDOC [18]. COBRA distinguishes between Temporal Entities (see Section 3.1) and *Persistent Entities* which are disjoint. This terminology is borrowed from CIDOC [18] but is conceptually inline with DOLCE [17], whereby Endurant corresponds to Persistent Entity and Perdurant to Temporal Entity. In short, Temporal Entities are entities that have a temporal extent whereas Persistent Entities are essentially independent of time. COBRA uses this high-level categorisation as a foundational basis but it doesn't go however much further in the reuse of existing foundational ontologies for it aims at supporting analysis of processes and a complete grounding into this kind of ontologies would carry an important computational overhead. Instead, we provide a simple categorisation of Persistent Entities specifically tailored to our needs, though informed by DOLCE, whereby we contemplate Physical and Non-Physical Entities which are disjoint. Physical entities are those that have a mass.

Physical and Non-physical Entities are further refined into *Agentive* and *Non-Agentive*. The distinction between these classes which are obviously disjoint, is that Agentive Entities are those that can take an active part within some specific activity. Finally, we define *Agent* as the union of both Physical and Agentive Non-Physical Entities. We include for reference and self-containment a few concepts widely used within BPM. For instance, we include *Object*, *Person*, *Organisation*, *Software Agent*, and *Role*. The latter will be dealt with in more detail later on. COBRA, for its purpose is to provide core definitions for supporting business analysis, does not refine these classes any further. Instead they serve as placeholders for including additional conceptualisations such as Organisational Ontologies or domain-specific master data. By doing so we aim at reducing the ontological commitment, while we support the seamless integration of further specific conceptualisations. Finally, since sometimes one needs not specify a concrete instance but rather the type, e.g. "you require a computer", we have defined the meta-class Persistent Entity Type such that all the sub-classes of Persistent Entity are instances of Persistent Entity Type. This is depicted in Figure 2 by means of a double-headed arrow.

Core concepts in COBRA are *Business Activity* and *Business Activity Realisation*. A Business Activity is a Non-Agentive Non-Physical Entity (the isA relation is not depicted in the figure for the sake of clarity) that represents the specification of any business activity at a high-level where aspects such as the control flow are abstracted away. We contemplate two kinds of Business Activ-

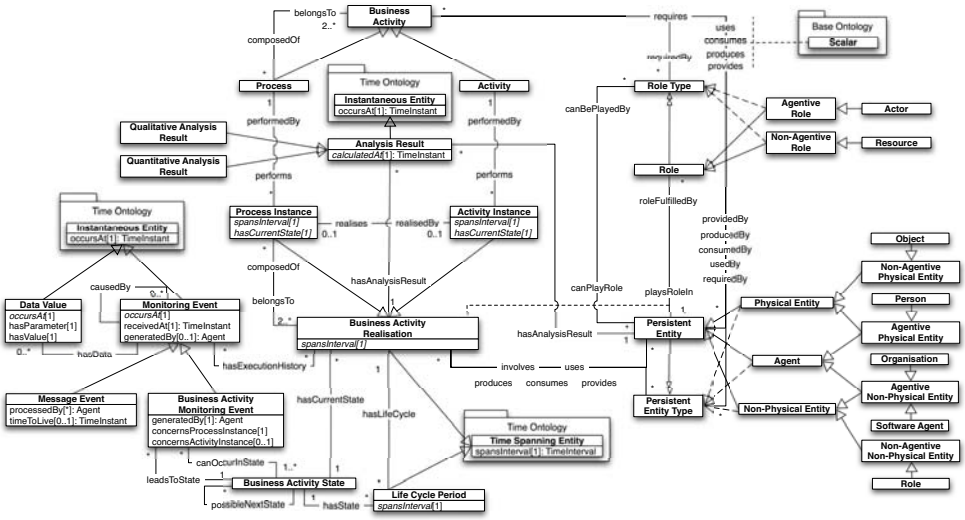


Fig. 2. Core Business Process Analysis Ontology.

ities, namely *Process* and *Activity*, to reuse the terminology typically employed in the BPM domain [12]. Activity represents atomic Business Activities whereas Processes are *composedOf* at least two other Business Activities. Business Activity Realisations are Time Spanning Entities which represent the actual execution of Business Activities. Mirroring Business Activities, Process Instance and Activity Instance are the two kinds of Business Activity Realisations considered. Despite their name, which originates again from BPM literature, both are concepts which represent the actual executions of Processes and Activities respectively (see *performs* in Figure 2). In this way it is simple to move between fine-grained details concerning one single execution and aggregation details concerning all the executions of the same Business Activity. Additionally, we include the relation *realises* between Process Instance and Activity Instance in order to track the fact that what appears as an Activity for some Process might in fact be supported by a complex Process.

COBRA primarily characterises Business Activities from the perspective of the Persistent Entities involved since we aim to cover the Resource and Object views typically adopted in BPA. Our approach is based on the notion of *Role*[4]. Role, much like in the Descriptions and Situations ontology [19], is the function assumed or part played by a Persistent Entity in a particular Business Activity Realisation. This is defined by means of the ternary relation *playsRoleIn* which relates Roles, Persistent Entities and Business Activity Realisations. COBRA includes a simple categorisation of Roles into two disjoint types, *Agentive* and *Non-Agentive* ones. Agentive Roles are those that can only be played by Agents whereas Non-Agentive Roles can, in principle, be played by any Persistent Entity. Further restrictions should be defined on a per Role basis. COBRA currently includes for self-containment an Agentive Role–*Actor*–and a Non-Agentive Role–

[4] Role is duplicated in the figure for the sake of clarity

Resource–which are of most use when analysing business processes. Again, Roles categorisation is to be extended for specific domains. Finally, we include the *Role Type* meta-class in order to support describing things like "we require an engineer". Persistent Entities are further characterised by a set of Role Types they can play within Business Activity Realisations. This allows to model for example that "Directors can play the Role Type Supervisor".

Given the notion of Role and how these relate to Persistent Entities, we can now fully describe Business Activity and Business Activity Realisation. Business Activities may *use*, *consume*, *produce*, and *provide* a set of Persistent Entity Types. The relationship *uses* may also be defined over specific Persistent Entities, e.g., "this Activity requires this specific machine", reason why we actually include two relations *usesPersistentEntity* and *usesPersistentEntityType*. Usage, like in the Enterprise Ontology, concerns Persistent Entities that can play a Resource Role, and which are not consumed during the execution of the business activity. In other words, the availability of the Resource will decrease during the execution of the Business Activity and will be restored to the original level at the end. For example, we use a screw-driver for screwing but as soon as we are done the screw-driver is available for others to use. Resource consumption is captured by means of the relationship *consumes*. This relationship is only applicable to those Persistent Entity Types which are not Agents. For situations where some things are required but not used or consumed, we provide the relation *requires*. Business Activities may *require* a set of Persistent Entities (e.g. "a particular document is required"), Persistent Entity Types (e.g. "one license is required to use the software"), and Role Types (e.g. "a coordinator is required") in order to be performed. The three scenarios are modelled as separate relations. The relationship *produces* captures the outcomes of a Business Activity and is applicable to Persistent Entity Types excepting Non-Agentive Non-Physical Entities for which we have devoted instead the relationship *provides*. These two relationships allow us to capture things like "this production process produces a robot" and "market analysis provides knowledge". These, excepting the relationship *provides*, are all ternary relationships that can be characterised by the quantity involved, see dashed line in Figure 2.

When it comes to Business Activity Realisations we capture their relation with Persistent Entities in a very similar manner. We do so by means of five relations–*involves*, *uses*, *consumes*, *produces*, and *provides*. Whereby *involves* is a super-relation of the others. We finally provide a ternary relation between Business Activity Realisation, Persistent Entity, and Role which allows us to capture the Role a Persistent Entity *plays in* a Business Activity Realisation (see *playsRoleIn* in Figure 2). Business Activity Realisations are the bridge between the high-level conceptualisation of the BPM domain and the low-level monitoring information captured at runtime by the IT infrastructure. Thus, Business Activity Realisations are further characterised by an *execution history*, a *life-cycle*, and the *current state* of the execution.

The execution history is a set of *Monitoring Events* relevant for monitoring the life-cycle of a Business Activity, see Figure 2. Monitoring Events are

Instantaneous Entities generated by Agents. They are characterised by a reception timestamp which is to be filled by the logging infrastructure upon reception of an event. The main goal of this attribute is to support monitoring even in environments where clock synchronisation mechanisms are hardly applicable. Additionally, Monitoring Events can have a causality vector, i.e., the set of Monitoring Events that caused that particular event. This supports capturing the actual derivation of events by the monitoring infrastructure as necessary for Complex Event Processing. Finally, Monitoring Events might be characterised by additional associated data which is expressed as *Data Value* instances. These instances identify a particular parameter and the value associated to it.

Monitoring Events are further refined into *Message Events* and *Business Activity Monitoring Events*. The former accommodates Event-Based environments so that their execution can also be traced. The latter supports monitoring the life-cycle of Business Activity Realisations in Process-Aware Information Systems. Business Activity Monitoring Events therefore *concern* a specific Process Instance and, depending on the granularity of the event occurred, may also concern an Activity Instance. Similarly to the proposals in [20, 12, 19], Business Activity Monitoring Events are centred around the notion of state model. Every event identifies a particular transition within the state model, the transition being indicated by means of the *leadsToState* attribute. Conversely the canOccurInState attribute allows to ensure that the transitions are consistent with the prescribed state model or to detect anomalies within the execution history possibly due to missing events.

COBRA supports the definition of specific state models is a simple ontological form by means of the *Business Activity State* concept which has a set of *possibleNextStates*. Business Activity States are used to further characterise Business Activity Realisations with the *hasLifeCycle* and *hasCurrentState* slots. The former captures the overall life-cycle of Business Activity Realisations as a set of Life-Cycle Periods which are Time Spanning Entities whereby the executed business activity was in a particular state. The latter is a shortcut for avoiding heavy usage of temporal reasoning in order to obtain the current state. On the basis of these Life-Cycle Periods it is possible to revisit the complete life-cycle of a Business Activity Realisation in a suitable manner for interval-based temporal reasoning. Instead of prescribing a particular state model and the corresponding events COBRA remains agnostic from the domain-specific details. Still, we provide an extension, i.e., Events Analysis Ontology, with a set of generic event processing forward-chaining rules that can derive information based Business Activity Monitoring Events. These rules will be detailed in the next section.

Finally, given that COBRA aims to support Business Process Analysis, both Persistent Entities and Business Activity Realisations are characterised by a set of *Analysis Results*. Thus one can capture results of previous analysis for all the relevant perspectives for BPA. Analysis Results are Instantaneous Entities of a *Quantitative* or *Qualitative* nature[5]. Being part of the core ontology for analysing business process, this allow us to reuse results across different types

[5] Note that we have used slot renaming for occursAt

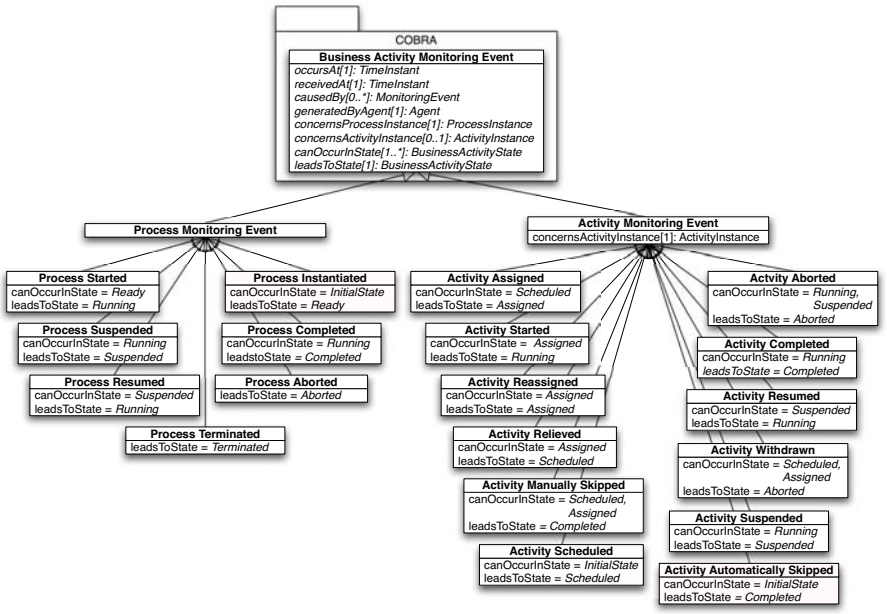


Fig. 3. Events Ontology.

of BPA which paves the way for enhancing current analysis techniques [11]. For instance, metrics computed at runtime can be reused when performing RBE, mining results can be applied during monitoring, etc.

4 Events Processing

COBRA aims at providing a conceptual and extensible framework for supporting BPA. Thus, it purposely leaves many aspects, such as domain-specific data or infrastructure specific monitoring events unspecified. In order to apply COBRA to specific domains these particular details have to be modelled and integrated. As part of the overall BPA conceptual framework but also in order to validate and test our conceptualisation we have developed an ontology for capturing monitoring events from a plethora of BPM systems and a general purpose Events Analysis Ontology that allows to derive information in terms of COBRA from monitoring information. In the remainder of this section we shall describe first the Events Ontology and next the Events Analysis Ontology.

4.1 Events Ontology

BPA takes the audit trails generated by the supporting IT infrastructure as a starting point, and attempts to derive information from the business perspective. Each of the supporting systems provides its own level of detail, in heterogeneous formats making it particularly difficult to integrate the audit trails generated as well as it complicates the creation of general purpose solutions. Common formats have been proposed as a solution to overcome this problem, e.g., MXML [20]

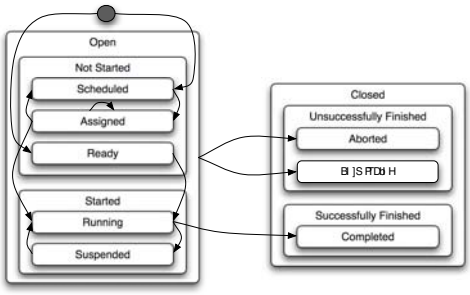


Fig. 4. Events Ontology State Model.

or the Audit Trail Format by the Workflow Management Coalition (WFMC). Although these formats have proven their benefits, they are supported by technologies that are not suitable for automated reasoning. In order to overcome this, we have extended COBRA with a reference Events Ontology (EVO) that provides a set of definitions suitable to a large variety of systems and ready to be integrated within our core ontology for analysing business processes. EVO is however an optional module which can be replaced by other models if required.

EVO is based on the previously mentioned formats since they provide general purpose solutions that have shown to be suitable to capture logs generated by a plethora of systems. As prescribed by COBRA, EVO is centred around a state model that accounts for the status of processes and activities, see Figure 4. The figure shows the different states and possible transitions contemplated for both Process Instances and Activity Instances which we believe are self-explaining. Note that process abortion differs from process termination in that in the former any ongoing activity is allowed to finish [12]. The dark dot represents the initial state, arrows represent transitions, the smaller boxes depict states, and bigger boxes encapsulate (conceptual) families of states. The state model has been captured ontologically as shown in Figure 3, an enhanced with additional relations. For instance it is possible to determine whether an Activity Instance has been allocated–*isAllocated*–which is true for those that are either in state Running, Suspended, or Assigned. It is also possible to determine whether a Business Activity Realisation is active–*isActive*–which is equivalent to Running, or inactive–*isInactive*–which is true for the rest of the states.

The state model does not distinguish between Process Instances and Activity Instance. The reason for this is mainly to simplify some tasks, e.g. monitoring of active Business Activity Realisations. Still, this necessary distinction is preserved within the logs by means of the Business Activity Monitoring Events defined, see Figure 3. EVO includes two subclasses, namely *Process Monitoring Event* and *Activity Monitoring Event*. EVO currently captures seven Process Monitoring Events and twelve Activity Monitoring Events based on the state model in Figure 4. Process Monitoring Events capture the different transitions which are possible for Process Instances. A Process Instance can be *Instantiated*, *Started*, *Suspended*, *Resumed*, *Completed*, *Aborted* and *Terminated*. Activity Monitoring Events, in addition to the typical execution events, contemplate the distribution of work to Agents. Thus, there are events that capture the scheduling of activ-

ities, the *Assignment*, *ReAssignment*, or *Relief* of activities to specific agents. Additionally like MXML, EVO contemplates the possibility for skipping activities either manually or automatically, which lead to a correct completion. Finally, EVO captures the abortion of activities by means of two events *Activity Aborted* and *Activity Withdrawn*. The distinction between the two lays in the fact that only started activities can be aborted.

4.2 Event Analysis Ontology

So far we have focussed on the conceptual models that capture the BPM domain spanning from the low-level details concerning audit trail information, to higher-level aspects such the roles played by agents in certain processes. In this section we focus on how, on the basis of this conceptual model and by capturing monitoring information ontologically, we derive knowledge about the enterprise that can then support business practitioners or even Knowledge-Based Systems in the analysis and eventual decision-making process.

OCML provides support for defining both backward and forward-chaining rules. In order to derive information upon reception of monitoring events we have defined a set of generic forward-chaining rules which are independent from the domain and the specific Monitoring Events defined. The goal is to provide reusable rules which can then be enhanced with domain specific ones to derive a richer knowledge-base. Additionally we have implemented a set of relations which are of most use when analysing processes. Some of these relations have been defined for COBRA in a generic manner, whereas others have been bundled with EVO for they are EVO-specific. The rules currently implemented support (i) deriving and checking the consistency of life-cycle information about Business Activity Realisations; (ii) updating the execution history of Business Activity Realisations; (iii) updating the relations between Process Instances and Activity Instances; (iv) tracking the Agents involved and; (v) updating the Roles played by Actors within Business Activities.

The first set of rules uses Business Activity Monitoring Events to update the current state of activity realisations, generate Life-Cycle Period instances, and contrast the transitions with the given state model. Basically, every event defines the end of a period and the beginning of a new one[6]. In this way, by simple updates over the life-cycle and with temporal reasoning we can support many of the monitoring competency questions previously exposed. To this end we provide a general purpose relation that holds when, given a Business Activity Realisation, a Time Instant, and a Business Activity State, the activity realisation was in the state given at that particular instant.

The second set of rules aim at correctly tracking the execution history for specific Business Activities so that they can later on be used within Business Process Mining algorithms. The third aspect is supported by a rule that tracks the coincidence of Process Instances and Activity instances within the same Business Activity Monitoring Event and derives the appropriate *composedOf*

[6] The initial state is a special one which is predefined in COBRA

relation. Agents involvement is derived from the *generatedBy* slot in the events. Finally, whenever one of the Actors involved is the only one taking part in a Business Activity Realisation that can play a certain Role Type that was required, we can derive the role this Actor played. This last rule is bundled with EVO since it is necessary to know whether the business activity was completed before deriving this. The interested reader is referred to the ontologies for further details about the rules and relations currently implemented.

5 Conclusions and Future Work

BPM systems aim at supporting the whole life-cycle of business processes. However, BPM has made more evident the current lack of automation that would support a smooth transition between the business world and the IT world. Yet, moving back and forth between these two aspects is a bottleneck that reduces the capability of enterprise to adapt to ever changing business scenarios. As a consequence there is a growing need for integrating semantics within the BPM domain. A crucial branch of BPM where semantics have a clear and direct impact is Business Process Analysis, where in fact so-called Business Intelligence is appearing as a key enabler for increasing value and performance [3]. Important efforts but with limited success have been devoted to integrating semantics within BPA. The reason for this appears to be the fundamental gap between semantics technologies and the ones currently deployed within BPM solutions.

To reduce this gap we have defined COBRA, a core ontology for business process analysis. The research carried has been guided on a set of competency questions extracted from existing needs with the BPA domain. Our approach is based on a conceptualisation that links low-level monitoring details with high-level business aspects so as to bring this vital information to the business-level as required by business practitioners. This conceptual model is based on a Time Ontology and has been enhanced and validated by means of two extensions for logging monitoring information in a semantic manner, and eventually processing this information.

A key requirement underlying our work has been the need to produce a generic yet comprehensive conceptual infrastructure where additional extensions can be seamlessly plugged-in in order to better support BPA techniques. Future work will thus be devoted to extending our work along the vision previously presented in [11]. In particular, next steps will be devoted to the definition of a metrics computation engine that will support the computation of both generic and user defined metrics, and the implementation of a classification Problem-Solving Method for detecting process deviations. In parallel, we are working on an ontology-based user interface to the reasoning infrastructure as part of WSMO Studio (www.wsmostudio.org).

References

1. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., de Medeiros, A.K.A., Song, M., Verbeek, H.M.W.: Business process mining: An

industrial application. Information Systems **32**(5) (2007) 713–732
2. Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel, D.: Semantic business process management: A vision towards using semantic web services for business process management. In Lau, F.C.M., Lei, H., Meng, X., Wang, M., eds.: ICEBE, IEEE Computer Society (2005) 535–540
3. Watson, H.J., Wixom, B.H.: The current state of business intelligence. Computer **40**(9) (2007) 96–99
4. Fox, M.S.: The tove project towards a common-sense model of the enterprise. In: IEA/AIE '92: Proceedings of the 5th international conference on Industrial and engineering applications of artificial intelligence and expert systems, London, UK, Springer-Verlag (1992) 25–34
5. Uschold, M., King, M., Moralee, S., Zorgios, Y.: The enterprise ontology. Knowledge Engineering Review **13**(1) (1998) 31–89
6. Geerts, G.L., McCarthy, W.E.: An accounting object infrastructure for knowledge-based enterprise models. IEEE Intelligent Systems **14**(4) (1999) 89–94
7. Gordijn, J., Akkermans, H.: Designing and evaluating e-business models. IEEE Intelligent Systems **16**(4) (2001) 11–17
8. Malone, T.W., Crowston, K., Herman, G.A.: Organizing Business Knowledge: The MIT Process Handbook. MIT Press, Cambridge, MA, USA (2003)
9. Castellanos, M., Casati, F., Dayal, U., Shan, M.C.: A comprehensive and automated approach to intelligent business processes execution analysis. Distributed and Parallel Databases **16**(3) (2004) 239–273
10. Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., Shan, M.C.: Business process intelligence. Computers in Industry **53**(3) (2004) 321–343
11. de Medeiros, A.K.A., Pedrinaci, C., van der Aalst, W.M.P., Domingue, J., Song, M., Rozinat, A., Norton, B., Cabral, L.: An Outlook on Semantic Business Process Mining and Monitoring. In: Proceedings of International IFIP Workshop On Semantic Web & Web Semantics (SWWS 2007). (2007)
12. zur Muehlen, M.: Workflow-based Process Controlling. Foundation, Design, and Implementation of Workflow-driven Process Information Systems. Volume 6 of Advances in Information Systems and Management Science. Logos, Berlin (2004)
13. Motta, E.: Reusable Components for Knowledge Modelling. Case Studies in Parametric Design Problem Solving. Volume 53 of Frontiers in Artificial Intelligence and Applications. IOS Press (1999)
14. Hobbs, J.R., Pan, F.: Time ontology in owl. Available at http://www.w3.org/TR/owl-time/ (2006)
15. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**(11) (1983) 832–843
16. Vilain, M.B.: A system for reasoning about time. In: AAAI. (1982) 197–201
17. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., Schneider, L.: WonderWeb Deliverable D17. The WonderWeb Library of Foundational Ontologies and the DOLCE ontology. http://www.loa-cnr.it/Papers/DOLCE2.1-FOL.pdf (2003)
18. ICOM/CIDOC CRM Special Interest Group: CIDOC Conceptual Reference Model. http://cidoc.ics.forth.gr/docs/cidoc_crm_version_4.2.2.pdf (2007)
19. Gangemi, A., Borgo, S., Catenacci, C., Lehmann, J.: Task taxonomies for knowledge content. Technical report, EU 6FP METOKIS Project D07 (2004)
20. van Dongen, B., de Medeiros, A., Verbeek, H., Weijters, A., van der Aalst, W.: The prom framework: A new era in process mining tool support. In: Applications and Theory of Petri Nets 2005. 26th International Conference, ICATPN 2005, Miami, USA, Springer-Verlag (2005) 444–454