PoSecCo

Policy and Security Configuration Management

Project Title	Policy and Security Configuration Management
Project Acronym	PoSecCo
Project No	257109
Instrument	Integrated Project
Thematic Priority	Information and Communication Technologies
Start Date of Project	01.10.2010
Duration of Project	36 Months
Project Website	`http://www.posecco.eu`

D4.3 – TAILORING SEMANTIC PROCESS MINING METHODS TO BEHAVIORAL LANDSCAPE MODELS

Work Package	WP4, Operational Landscape Description & Audit
Lead Author (Org)	Jan Martijn van der Werf (TUE)
Contributing Author(s) (Org)	Wil van der Aalst (TUE), Annett Laube (BFH), Eric Verbeek (TUE)
Due Date	30.09.2011
Date	03.10.2011
Version	1.0

Dissemination level

X	PU: Public
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

PoSecCo project (project no 257109) is partially supported/co-funded by the European Community/ European Union/EU under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme for R&D (FP7). This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

PoSecCo

Versioning and contribution history

Version	Date	Author	Notes
0.1	19.07.2011	Jan Martijn van der Werf (TUE)	Draft version
0.2	12.09.2011	Annett Laube (BFH)	Draft Exec Summary and Conclusion
0.3	13.09.2011	Jan Martijn van der Werf (TUE)	Incorporated comments Serena Ponta
0.4	29.09.2011	Jan Martijn van der Werf (TUE)	Incorporated comments internal review Günther Karjoth
1.0	03.10.2011	Jan Martijn van der Werf (TUE)	Approved by Ronald Maier (UIBK)

Disclaimer

CONTENTS

EXECUTIVE SUMMARY

Future Internet (FI) applications are compositions of services of different kinds and layers, ranging from low-level infrastructure services to high-level business services. These services communicate in order to deliver a business service. Communication in FI applications is mostly asynchronous: services communicate via message exchange.

The PoSecCo functional system model describes the structure of FI applications in three layers: the business layer, the IT layer and the infrastructure layer (as described in more detail in Deliverable D4.2). The business layer describes the business services that a service provider offers, and which business processes implement these services. The IT layer describes the IT service compositions that implement each of the business processes, and the actual implementation of these service compositions and their deployment is described in the infrastructure layer. Such a structural description of the service landscape does not consider the interactions between services, i.e., the choreography, at any architectural level.

Behavior, i.e., the order in which messages are sent and received, can exist on the different architectural levels (Business, IT Layer, Infrastructure) and can be described using languages like Business Process Modeling Notation (BPMN) or the Web Services Choreography Description Language (WS-CDL). On the Business Layer, behavioral aspects describe which business (sub-)processes interact and in which order the business processes are executed. On the IT Layer, the main behavior lies in the communication between IT resources, focusing on the order in which messages are sent, like software components composing a given IT service. The Infrastructure layer describes the implemented soft- and hardware realizing the different abstract IT resources. Once deployed, the software components generate execution data to enable the monitoring of the service landscape.

Process mining techniques use these execution data in the form of event logs to analyze the behavior of the landscape components. In process mining, we can distinguish three main activities:

- Discovery: From an event log a process model, e.g. a Petri net, is generated without using any a-priori information.
- Conformance: An existing model is compared with an event log of the system. It is checked if reality, as recorded in the log, conforms to the model and vice versa.
- Enhancement: An a-priori model is changed or extended using the data from event logs.

All three process mining activities can be used within PoSecCo. Discovery and enhancement can mainly be used to discover and update behavioral aspects in the different model layers of the service landscape, whereas conformance can be used to check behavioral aspects during audit activities of service offerings of service providers.

Semantic process mining techniques mainly focus on conformance. Many specifications a system execution should adhere to are expressed in high-level terms at the business layer, whereas the execution data resides on the lowest level at the infrastructure. In order to automatically check these high-level specifications, we need to bridge the gap between the different layers of abstraction. To do so, we need to relate elements in the event logs to concepts at higher levels of abstractions. For this, we combine techniques from the semantic web, like ontologies with process mining techniques. These techniques allow us to combine elements from different information sources, and to use reasoning to verify the high level specifications.

The PoSecCo functional system model can be translated into an ontology (cf. Deliverable D3.2), and then be used in semantic process mining as an additional source of information. In this way, it is possible to define high-level policies on the business layer, and verify these policies on the event logs available in the infrastructure layer.

1 INTRODUCTION

1.1 Purpose and Scope

This deliverable describes behavioral aspects within the architecture of Future Internet (FI) applications in the scope of PoSecCo. Whereas PoSecCo defines a policy design methodology to design a *golden configuration* of the service landscape top-down starting with requirement engineering of business processes, process mining allows the analysis of the AS-IS situation of an existing service landscape. Figure 1 depicts the relation between the PoSecCo architecture tools (introduced in Deliverable D1.2) and process mining.

The *golden configuration* defines an ideal set of configurations that complies with all security requirements and that implements these in a cost-efficient way. This *golden configuration* is used to configure the service landscape. A configuration management system (CMS) is then used to configure the real service landscape according to the golden configuration. In PoSecCo, we assume that the CMS is aligned with the actual, the real configuration of the service landscape.

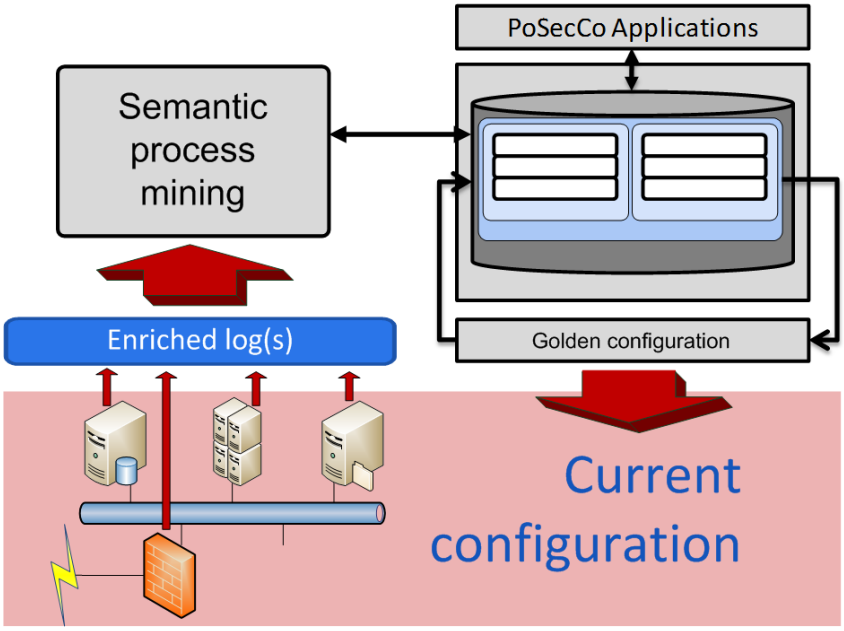

Figure 1: Process mining in PoSecCo

In the real service landscape with the *current configuration*, that can be retrieved from the Configuration Management Database (CMDB), execution logs are generated. Based on these logs, and the information available from the PoSecCo Model Repository, process mining aims at the analysis of behavioral aspects of such service landscapes. By analyzing the AS-IS situation, process mining can also help to identify the elements present in a service landscape and to validate and update the model in the PoSecCo Model Repository.

Semantic process mining techniques use in addition to the event log other sources of information to analyze behavioral aspects of a service landscape. This allows not only to discover behavioral aspects in the different architectural layers, but also to check the conformance of these models to rules or policies.

This deliverable is intended to explore the possible behavioral aspects within a service landscape, and how process mining can be used in PoSecCo. Deliverable D4.7 will extend the PoSecCo functional system model with behavioral aspects.

1.2 Structure of the Document

This deliverable is structured as follows. In Section 2, we define the concepts of behavior and processes, and explore the PoSecCo functional system model (described in detail in Deliverable D4.2) discussing where these concepts can be found in the PoSecCo functional system model. The abstract descriptions are accompanied by real-life examples from our use case partner Crossgate. Section 3 discusses process mining in general and describes the three main activities in process mining – discovery, conformance and enhancement – in more detail. In Section 4, we present semantic process mining in relation with the PoSecCo project. Section 5 concludes this deliverable.

2 BEHAVIOR IN FI APPLICATIONS

The design of FI applications comprises both the structure of the services, called their *composition*, as well as their dynamics, called its *behavior*. Dynamic aspects that need to be considered during the design of such applications is the order in which services are invoked, and how these services communicate. Communication in FI applications is asynchronous: services communicate via message exchange. The asynchronous nature of FI applications makes the dynamic aspects of FI applications error-prone: as messages can be received and processed in any order, such applications are very sensitive for undesired deadlocks. It therefore requires a well-considered behavioral design.

In this chapter, we introduce the basic concepts of modeling behavior using process models, and, based on a running example, indicate places in the service landscape where behavior is involved.

2.1 Process Models

A *process model* is a set of *tasks* with *causal dependencies* between these tasks. These causal dependencies define the *control flow* of the process model, i.e., the ordering in which the tasks can be executed. Like in basic programming, we can identify five basic patterns in causal dependencies:

- *Sequence*: a linear ordering between two tasks: task A is always directly followed by task B;
- *Or-split*: a choice has to be made between several outgoing branches;
- *And-split*: all outgoing branches will be executed;
- *Or-join*: at least one of the incoming branches should be ready in order to continue;
- *And-join*: all incoming branches should be ready in order to continue;

Many more advanced patterns exist for modeling the control flow. For a more elaborate overview on control flow patterns, we refer the reader to [6].

The execution of a process model is the execution of the tasks in that process such that its causal dependencies are met. To execute a task, *resources* can be used. Resources can be *consumable* or *durable*. Typical examples of consumable resources are parts, energy consumption or costs. Human operators and hardware are examples of durable resources.

An important aspect of process models is that it can be repeated: it can be executed for many different instances. An *instance* is the single execution of a process model, consisting of a state, which is determined

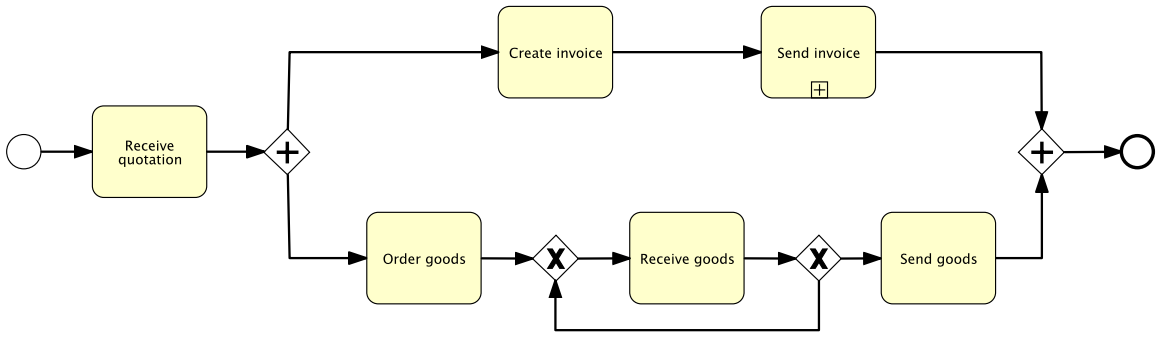

Figure 2: Example process model in BPMN 2.0 notation

by the tasks already executed on that instance. *Data* can be associated to each instance, and changed by executing tasks on the instance according to the control flow of the process model.

A process model defines two types of concurrency: within an instance, if two tasks can be executed simultaneously on the instance without influencing each other (e.g., no shared resources), and over instances, i.e., several instances of the same process model can be executed simultaneously, where only resources are shared over the different instances.

For modeling behavior, many different modeling languages exist. Formal languages like process algebras [20, 15] and Petri nets [26] and Linear Time Logic (LTL) [25] provide a formal ground for the verification of behavioral aspects within a model. Industrial languages include the Event-Process Chains (EPC) [28, 18] and Business Process Modeling Notation (BPMN) [22] for modeling business processes, the Business Process Execution Language for Web Services (BPEL4WS) [10] for modeling web services, and the Web Services Choreography Description Language (WS-CDL) [17] for modeling the interaction between web servies. A language can be *procedural* or *declarative*. In procedural languages all allowed behavior is modeled explicitly, e.g., process algebras, Petri nets, BPMN and WS-CDL, whereas in declarative languages, like LTL, the allowed behavior is left implicitly. Some languages have a graphical representation, like Petri nets and BPMN, others are only textual, like process algrebras, LTL and WS-CDL. An editor supporting many of these languages is Oryx [11].

An example process model in the BPMN 2.0 notation is depicted in Figure 2. This process model consists of five tasks, represented as rounded rectangles, one subprocess, represented as a rounded rectangle with a plus operator, and both an and-split and and-join, and an or-split and or-join. The process model models an example process when a quotation is received: after receiving a quotation from some customers, the goods on the quotation are ordered. After ordering the goods, at least once, but possibly multiple times, goods are received. When all goods all received, the goods are send to the customer. In the mean time, an invoice has been created and sent to the customer. Sending the invoice is a process in itself, and therefore represented as a subprocess.

2.2 Behavioral Aspects in the Landscape Model

In the previous section, we introduced the notion of behavior. In this section, we consider the functional system model used within PoSecCo and described in Deliverable D4.2. We indicate where in the model behavioral aspects can be found. For a more elaborate explanation of the functional system model, we refer the reader to Deliverable D4.2.

The functional system model serves as an architectural framework of a FI application. It describes the elements of the FI application and how these elements are related, covering the various aspects of such an application, from high level business aspects to low level infrastructural details like software and hardware components. Although the functional system model describes which elements are able to communicate, it does not describe how and in which order the elements communicate.

As described in Deliverable 4.2, the functional system model is a layered model with three layers: the business layer, the IT layer and the infrastructure layer. Based on a running example, behavioral aspects of each of the layers will be discussed.

Running Example

As a running example we will consider a service provider, CG, that offers the business service “EDI” to some customer C. The service implements the transportation of several types of EDI messages into a quotation for trading partner TP. If the EDI message is an invoice, it is signed by an external trusted party B. The service is implemented using a transaction engine on a small, private cluster, consisting of five virtual machines running on a network storage server and two high performance servers. The service has been designed using the PoSecCo tools (cf. Deliverable D1.2).

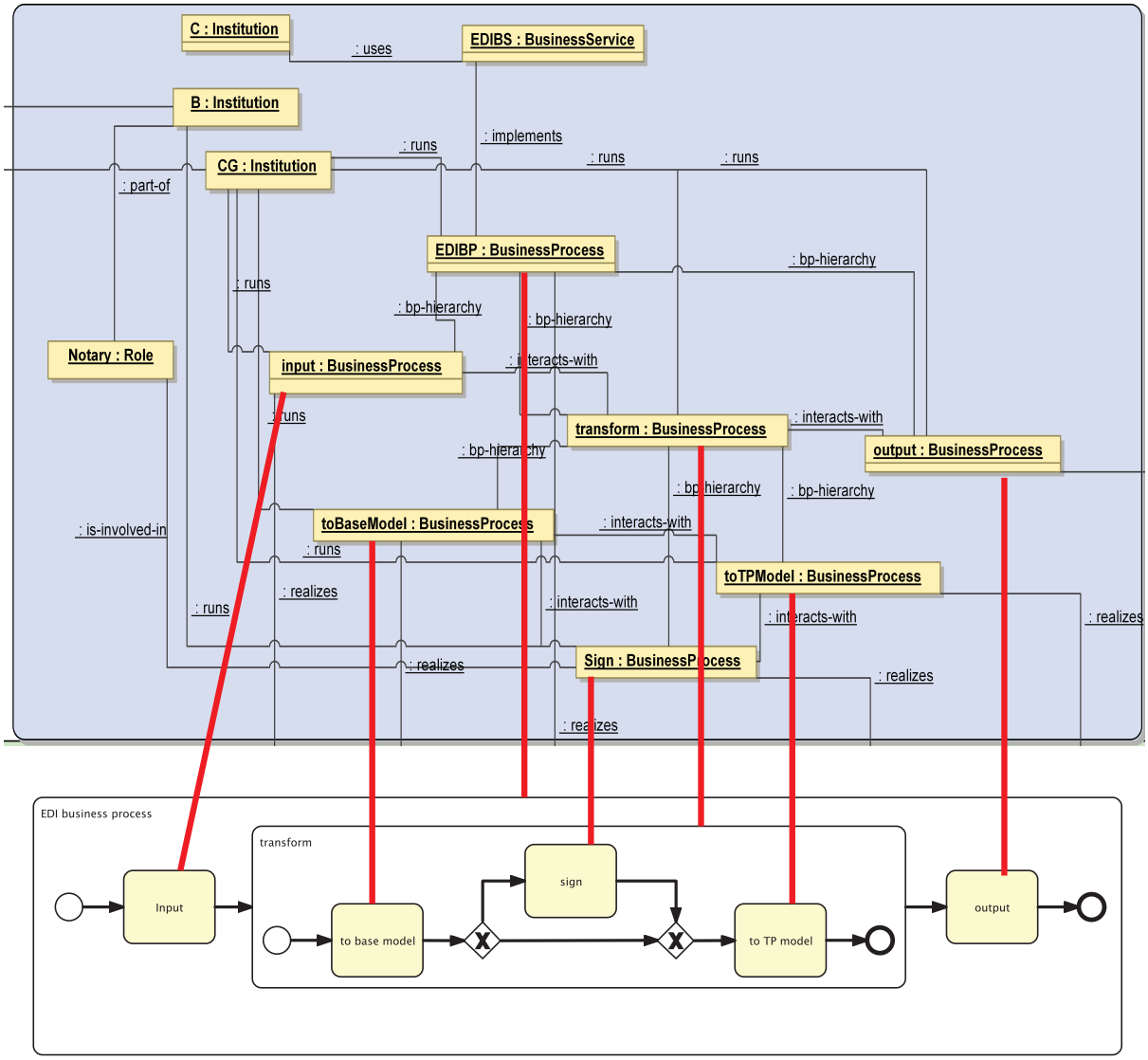

Figure 3: Business process of running example

2.2.1 Business Layer

The business layer of the functional system model involves the high level business services a service provider provides to its customers. Business services are implemented by a hierarchically structured set of business processes that interact. The functional system model covers the structure of these business processes: the subprocesses from which they are composed[1], and which business processes are allowed to interact. The behavioral aspect in the business layer describes how the interaction between business processes takes place.

Consider the running example. The business process implementing this service is designed hierarchically: the base business process implementing the EDI service is called “EDI business process”. It consists of two activities “input” and “output”, and one subprocess “transform”. The subprocess “transform” consists of three activities, “to base model”, “to TP model”, and “sign”. In the instance model, the structure, i.e., the hierarchy and the interaction, of the business process is stored, as shown in Figure 3. However, the order in which the activities occur, e.g., as depicted in Figure 3, is not stored in this model.

2.2.2 IT Layer

The IT layer defines the IT services and IT resources that implement the business processes defined in the business layer. For each of the business processes, an IT service model is created. This service model defines the IT resources with their interfaces, and how these interfaces are connected. These interfaces consist of operations that can be called. As in the business layer, the functional system model only describes the structure of which resources are allowed to communicate with which other resources. The order in which the resources are called is not specified.

To model the *interaction*, also called the *choreography* [2], between resources, i.e., the order in which the resources communicate, process modeling languages, like the Web Service Choreography Description Language (WS-CDL), or the newly defined Choreography view in BPMN 2.0 (see [22], Chapter 11) are used.

Consider again the running example. In this example, the business service is realized using different service models for the “EDI business process”, the “input business process”, the “output business process”, the “toBaseModel business process”, and the “toTPModel business process. Each of these service models is implemented and sold by CG. The “sign business process” is outsourced to company B. Figure 4 depicts the service models, the interfaces, and which components can communicate. Part of the corresponding instance model is shown in Figure 5. Although the business process model as depicted in Figure 3 implies the SAP Connector to be the initiator, this cannot be concluded from the structure in Figure 4 and Figure 5.

2.2.3 Infrastructure Layer

The infrastructure layer describes the configuration of the implemented landscape. Firstly, it defines which resources run on which nodes, and secondly, it defines the platform on which the resources run, like (virtual) servers and network peripherals. Whereas the behavior aspects on the other two layers describe the desired situation, this layer gives the current configuration of the different soft- and hardware components. The IT layer defines how and which resources communicate, and in which order. The infrastructure layer describes the real interactions.

Many different interactions exist on the infrastructure layer. For example, to monitor access control and the activities performed on a node, the node should log who accesses the machines, at what time, and what actions were performed. For example, a firewall should record when its management system has been accessed, and what changes have been performed. Moreover, a firewall should also record what traffic has been blocked, and what traffic has been allowed. In this way, it is possible to audit behavioral security aspects.

The nodes and resources within a service landscape should record their usage, like who accessed what

[1] Note that activities in a business process are represented as leaves in the hierarchical tree structure of business processes.

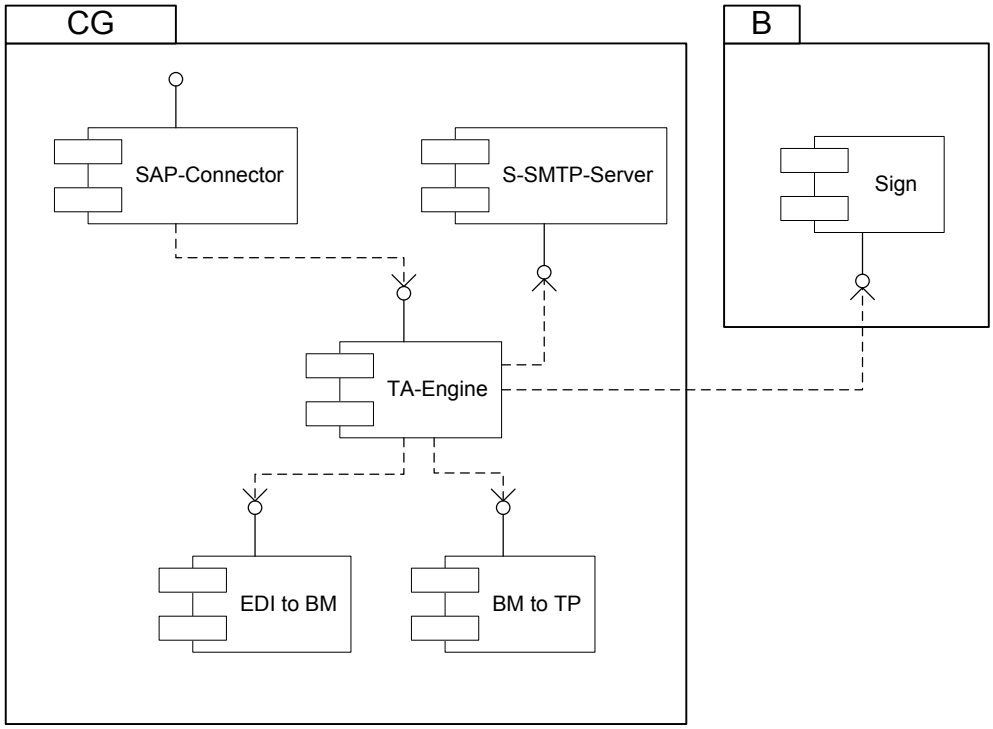

Figure 4: Service model of EDI service

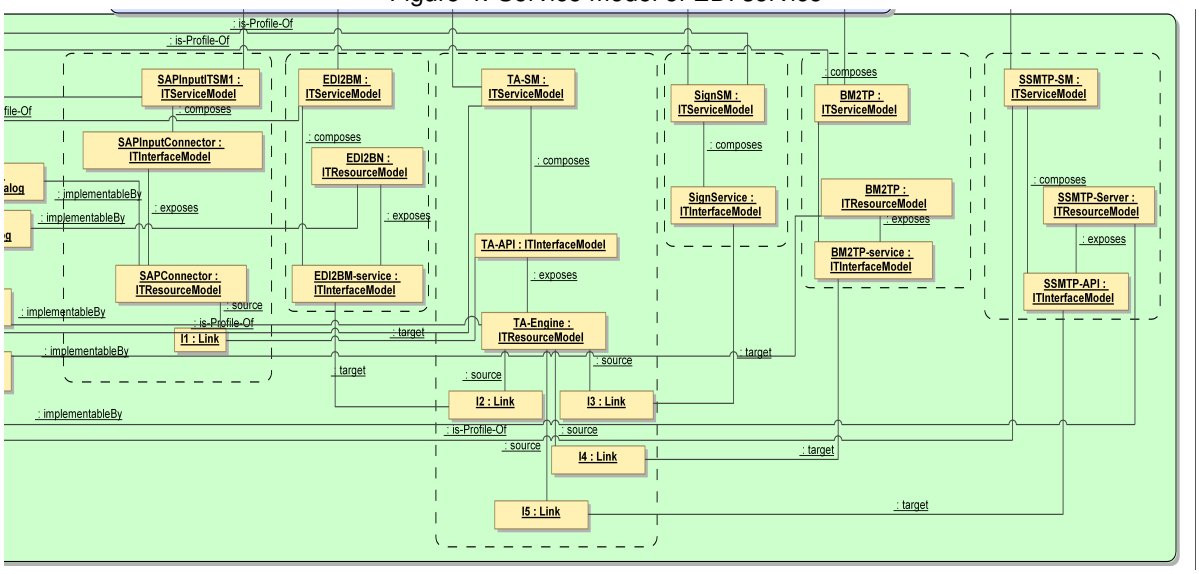

Figure 5: Part of the instance model for the running example

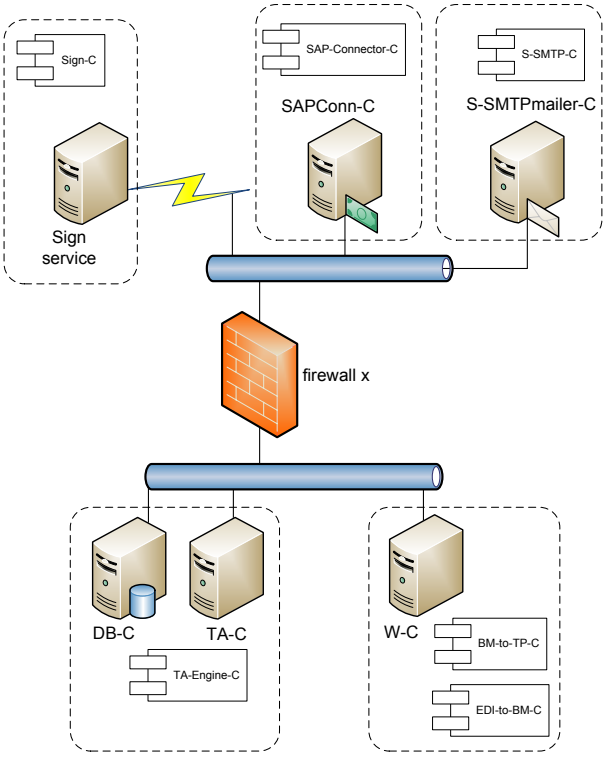

Figure 6: Infrastructure of running example

functionality, and when has which action been performed. In this way, a large data collection is created and maintained in the service landscape.

The ability to create event logs will be modeled in the Functional System Model (see Deliverable 4.2) as `logging` capability of the IT Layer concept `ITResource` that is the base class of all software installed in the systems. The `logging` capability relates the logging configuration settings to the instances of the `ITResource`. More information about capabilities and configurations will be found in the upcoming deliverable D3.3 - Configuration meta-model.

Consider again the running example. Figure 6 depicts the infrastructure. The two resources that need communication with the outside are placed before a firewall. There is a server, *SAPConn-C*, to run the SAPConnector resource *SAPConnector-C* specific for customer C, and server *S-SMTPmailer-C* running the customer specific resource *S-SMTP-C*. Behind the firewall, the transaction engine and transformation is running. The transaction engine runs on a single server named *TA-C*, and stores its data on the database server *DB-C*. Together, the servers serve the *TA-engine* resource. The last server, *W-C* runs the transformation services *BM-to-TP-C* and *EDI-to-BM-C*. The Sign service has been outsourced to the trusted party B. All resources are specific for customer C.

Each server needs to record when it has been accessed, by whom and what commands have been executed. Besides, the different resources require different levels of logging. For example, the transaction engine stores when which message has been send or received, and the server responsible for sending e-mails records when messages have been sent, and to whom.

The data recorded by the different nodes and resources in the infrastructure layer is the main input for process mining, which we will describe in the next sections.

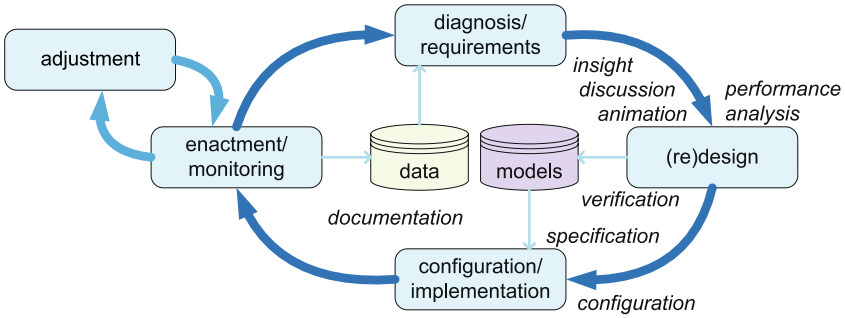

Figure 7: The BPM life cycle

3 PROCESS MINING

To position process mining [1], we first describe the so-called Business Process Management (BPM) life-cycle using Figure 7. The life-cycle describes the different phases of managing a particular business process. In the *design* phase, a process is designed. This model is transformed into a running system in the *configuration / implementation* phase. If the model is already in executable form and a Workflow Management (WFM) or BPM system is already running, this phase may be very short. However, if the model is informal and needs to be hardcoded in conventional software, this phase may take substantial time. After the system supports the designed processes, the *enactment/monitoring* phase starts. In this phase, the processes are running while being monitored by management to see if any changes are needed. Some of these changes are handled in the adjustment phase shown in Figure 7. In this phase, the process is not redesigned and no new software is created; only predefined controls are used to adapt or reconfigure the process. The *diagnosis/requirements* phase evaluates the process and monitors emerging requirements due to changes in the environment of the process (e.g., changing policies, laws, competition). Poor performance (e.g., inability to meet service levels) or new demands imposed by the environment may trigger a new iteration of the BPM lifecycle starting with the redesign phase.

As Figure 7 shows, process models play a dominant role in the (re)design and configuration/implementation phases, whereas data plays a dominant role in the enactment/monitoring and diagnosis/requirements phases. The figure also lists the different ways in which process models are used. Until recently, there were few connections between the data produced while executing the process and the actual process design. In fact, in most organizations the diagnosis/requirements phase is not supported in a systematic and continuous manner. Only severe problems or major external changes will trigger another iteration of the life-cycle, and factual information about the current process is not actively used in redesign decisions. Process mining offers the possibility to truly “close” the BPM life-cycle. Data recorded by information systems can be used to provide a better view on the actual processes, i.e., deviations can be analyzed and the quality of models can be improved.

Figure 8 shows that process mining establishes links between the actual processes and their data on the one hand and process models on the other hand.

Today’s information systems log enormous amounts of events. Classical WFM systems (e.g., Staffware and COSA), BPM systems (e.g.,BPM|one by Pallas Athena, SmartBPM by Pegasystems, FileNet, Global 360, and Teamwork by Lombardi Software), ERP systems (e.g., SAP Business Suite, Oracle E-Business Suite, and Microsoft Dynamics NAV), PDM systems (e.g., Windchill), CRM systems (e.g., Microsoft Dynamics CRM and SalesForce), middleware (e.g., IBM’s WebSphere and Cordys Business Operations Platform), and hospital information systems (e.g., Chipsoft and Siemens Soarian) provide detailed information about the activities that have been executed. Figure 8 refers to such data as event logs. All of the systems just mentioned directly provide such event logs.

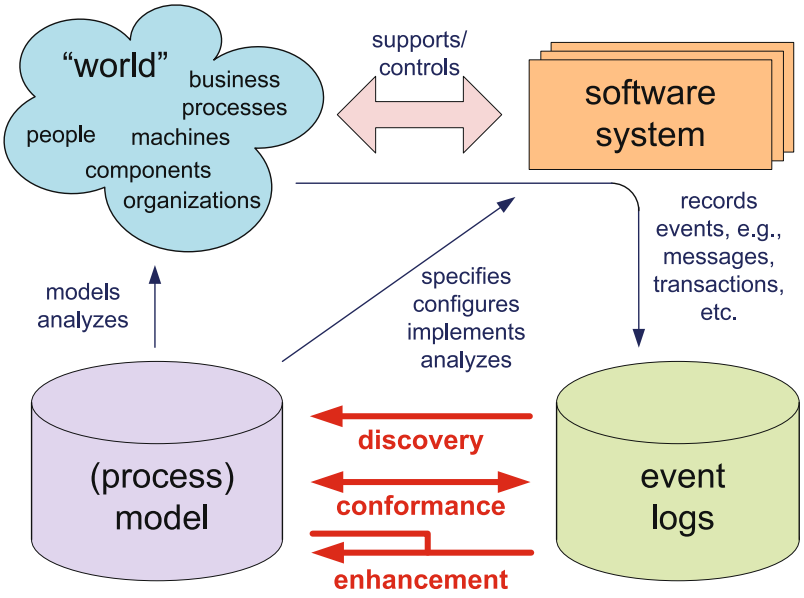

Figure 8: Software systems and its relation to process mining

However, most information systems store such information in unstructured form, e.g., event data is scattered over many tables or needs to be tapped off from subsystems exchanging messages. For example, Figure 9 shows possible execution data for system configured by the BPM diagram as shown in Figure 2.

Clearly, without any additional information it would be hard to link these four tables together in a meaningful way. For example, the third column of the history table could be a foreign key pointing to either one of the three supporting tables. Data retrieval and interpreting this data in the right way is one of the big challenges in process mining. For example, by examining the database schema, like in Figure **??**, the field names can be retrieved, and used to link the data. Or, if such a schema is not available, the system needs to be tested and examined in order to discover the right relationships. This step of *data extraction* is an integral part of any process mining effort.

As shown in Figure 8, we can identify three main activities in process mining: *discovery*, *conformance* and *enhancement*. Discovery focuses on re-creating models from event logs, conformance focuses on the question whether the system is executing according to some specification, and enhancement tries to enhance existing models using information from the event log. In the remainder of this section, we introduce event logs, and the three main activities in process mining. Most of the algorithms and techniques discussed in this chapter are implemented in the tool ProM [3], which can be found at `http://www.processmining.org`.

3.1 Event Logs

Table 1 shows a fragment of an event log that can be obtained from the execution data as shown in Figure 9. This table illustrates the typical information present in an event log used for process mining. We assume that an event log contains data related to a *single process*. Moreover, each event in the log needs to refer to a *single process instance*, often referred to as case. In Table 1, each request corresponds to a case, e.g., case "Derek W. Dick". We also assume that events can be related to some activity. In Table 1 events refer to activities "Receive quotation", "Create invoice", "Send invoice", "Order goods", "Receive goods", and "Send goods". These assumptions are quite natural in the context of process mining. All mainstream process modeling notations specify a process as a collection of activities such that the life-cycle of a single instance is described. Hence, the "Case" and "Activity" columns in Table 1 represent the bare minimum for process mining. Moreover, events within a case need to be ordered. For example, event "19651016" (the execution of

History				
19651016	9/8/10 10:25 AM	1	1	4
19651017	9/8/10 10:37 AM	1	2	6
19651018	9/8/10 1:19 PM	1	3	6
19651019	9/10/10 12:02 PM	2	1	3
19651020	9/11/10 1:40 PM	2	2	1
19651021	9/12/10 11:10 AM	1	4	5
19651022	9/12/10 2:17 PM	1	5	4
19651023	9/14/10 8:37 AM	2	4	2
19651024	9/14/10 11:38 AM	3	1	4
19651025	9/14/10 2:27 PM	3	4	5
19651026	9/16/10 9:28 AM	4	1	3
19651027	9/17/10 3:15 PM	4	4	2
19651028	9/18/10 1:27 PM	1	6	5
19651029	9/18/10 1:45 PM	4	5	3
19651030	9/18/10 2:03 PM	4	2	1
19651031	9/18/10 2:43 PM	5	1	4
19651032	9/18/10 3:10 PM	3	2	6
19651033	9/19/10 2:32 PM	3	5	4
19651034	9/20/10 1:06 PM	3	3	6
19651035	9/20/10 3:54 PM	5	4	5
19651036	9/20/10 4:32 PM	5	5	4
19651037	9/21/10 8:01 AM	5	6	5
19651038	9/21/10 8:37 AM	2	3	1
19651039	9/21/10 9:05 AM	4	6	2
19651040	9/21/10 9:10 AM	2	5	3
19651041	9/22/10 11:48 AM	5	2	6
19651042	9/22/10 2:24 PM	2	6	2
19651043	9/22/10 4:28 PM	5	3	6
19651044	9/23/10 9:42 AM	4	3	1
19651045	9/23/10 11:21 AM	3	6	5

Customer	
1	Derek W. Dick
2	Mark Kelly
3	Peter Trewavas
4	Steven Rothery
5	Ian Mosley

Activity	
1	Receive quotation
2	Create invoice
3	Send invoice
4	Order goods
5	Receive goods
6	Send goods

Agent	
1	Carol
2	Claire
3	John
4	Mike
5	Pete
6	Sue

Figure 9: Example execution data

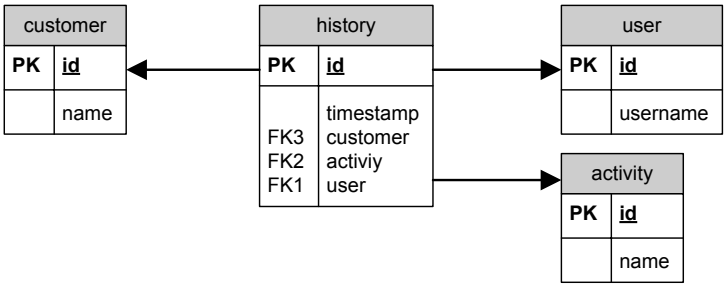

Figure 10: Possible data schema of execution data

activity “Receive quotation” request for case “Derek W. Dick”) occurs before event “19651021” (the execution of activity “Order goods” for the same case). Without ordering information, it is of course impossible to discover causal dependencies in process models.

Table 1 shows the information available per event. For example, all events have a timestamp (i.e., date and time information such as “9/14/10 8:37 AM”). This information is useful when analyzing performance related properties, e.g., the waiting time between two activities. The events in Table 1 also refer to resources, i.e., the persons executing the activities. In the context of process mining, these properties are called *attributes*.

Table 1: An example event log

Event	Timestamp	Case	Activity	Resource
19651016	9/8/10 10:25 AM	Derek W. Dick	Receive quotation	Mike
19651017	9/8/10 10:37 AM	Derek W. Dick	Create invoice	Sue
19651018	9/8/10 1:19 PM	Derek W. Dick	Send invoice	Sue
19651019	9/10/10 12:02 PM	Mark Kelly	Receive quotation	John
19651020	9/11/10 1:40 PM	Mark Kelly	Create invoice	Carol
19651021	9/12/10 11:10 AM	Derek W. Dick	Order goods	Pete
19651022	9/12/10 2:17 PM	Derek W. Dick	Receive goods	Mike
19651023	9/14/10 8:37 AM	Mark Kelly	Order goods	Claire
19651024	9/14/10 11:38 AM	Peter Trewavas	Receive quotation	Mike
19651025	9/14/10 2:27 PM	Peter Trewavas	Order goods	Pete
19651026	9/16/10 9:28 AM	Steven Rothery	Receive quotation	John
19651027	9/17/10 3:15 PM	Steven Rothery	Order goods	Claire
19651028	9/18/10 1:27 PM	Derek W. Dick	Send goods	Pete
19651029	9/18/10 1:45 PM	Steven Rothery	Receive goods	John
19651030	9/18/10 2:03 PM	Steven Rothery	Create invoice	Carol
19651031	9/18/10 2:43 PM	Ian Mosley	Receive quotation	Mike
19651032	9/18/10 3:10 PM	Peter Trewavas	Create invoice	Sue
19651033	9/19/10 2:32 PM	Peter Trewavas	Receive goods	Mike
19651034	9/20/10 1:06 PM	Peter Trewavas	Send invoice	Sue
19651035	9/20/10 3:54 PM	Ian Mosley	Order goods	Pete
19651036	9/20/10 4:32 PM	Ian Mosley	Receive goods	Mike
19651037	9/21/10 8:01 AM	Ian Mosley	Send goods	Pete
19651038	9/21/10 8:37 AM	Mark Kelly	Send invoice	Carol
19651039	9/21/10 9:05 AM	Steven Rothery	Send goods	Claire
19651040	9/21/10 9:10 AM	Mark Kelly	Receive goods	John
19651041	9/22/10 11:48 AM	Ian Mosley	Create invoice	Sue
19651042	9/22/10 2:24 PM	Mark Kelly	Send goods	Claire
19651043	9/22/10 4:28 PM	Ian Mosley	Send invoice	Sue
19651044	9/23/10 9:42 AM	Steven Rothery	Send invoice	Carol
19651045	9/23/10 11:21 AM	Peter Trewavas	Send goods	Pete

Until recently, the de facto standard for storing and exchanging events logs was *MXML* (Mining eXtensible Markup Language). MXML emerged in 2003 and was later adopted by the process mining tool ProM. Using MXML, it is possible to store event logs such as the one shown in Table 1 using an XML-based syntax. MXML has a standard notation for storing timestamps, resources, and transaction types. Moreover, one can add arbitrary data elements to events and cases. The latter resulted in ad-hoc extensions of MXML where certain data attributes were interpreted in a specific manner. For example, *SA-MXML* (Semantically Annotated Mining eXtensible Markup Language) is a semantic annotated version of the MXML format used by the ProM framework. SA-MXML incorporates references between elements in logs and concepts in ontologies. For example, a resource can have a reference to a concept in an ontology describing a hierarchy of roles, organizational entities, and positions. To realize these semantic annotations, existing XML elements were interpreted in a new manner. Other extensions were realized in a similar manner. Although this approach worked quite well in practice, the various ad-hoc extensions also revealed shortcomings of the MXML format. This triggered the development of *XES* (eXtensible Event Stream) [13].

XES is the successor of MXML. Based on many practical experiences with MXML, the XES format has been made less restrictive and truly extendible. In September 2010, the format was adopted by the *IEEE Task Force on Process Mining* [12]. The format is supported by tools such as ProM (as of version 6), Nitro, XESame, and OpenXES. See `www.xes-standard.org` for detailed information about the standard.

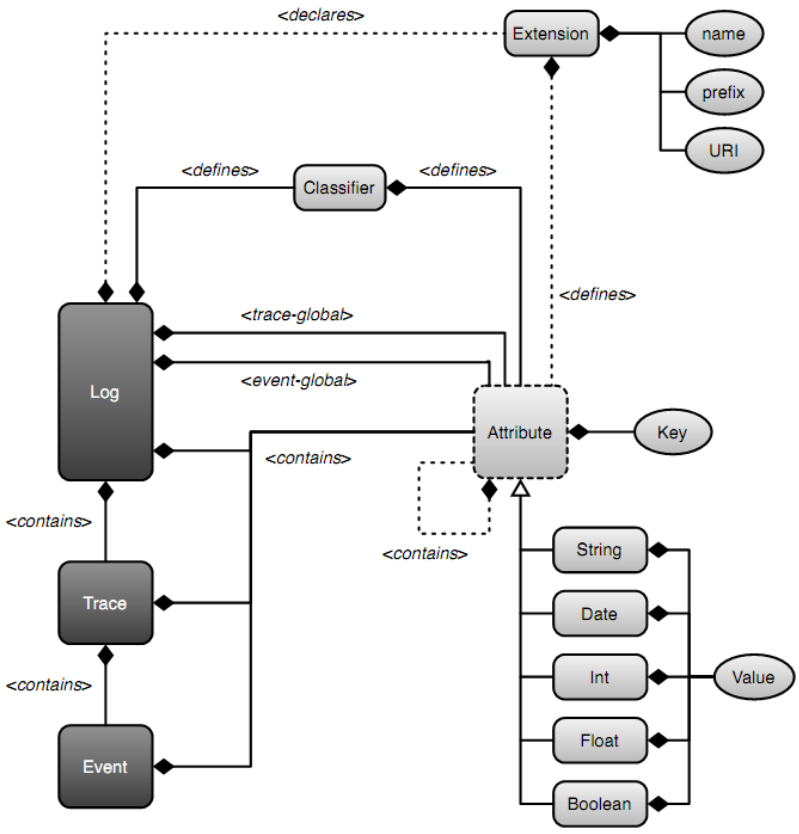

Figure 11: Meta model of XES

Figure 11 shows the XES meta model expressed in terms of a UML class diagram. An XES document (i.e., XML file) contains one log consisting of any number of traces. Each trace describes a sequential list of events corresponding to a particular case. The log, its traces, and its events may have any number of attributes. Attributes may be nested. There are five core types: *String*, *Date*, *Int*, *Float*, and *Boolean*. These correspond to the standard XML types: *xs:string*, *xs:dateTime*, *xs:long*, *xs:double*, and *xs:boolean*. For example, "2011-12-17T21:00:00.000+02:00" is a value of type *xs:dateTime* representing nine o'clock in the evening of December 17th 2011 in timezone GMT+2.

XES does not prescribe a fixed set of mandatory attributes for each element (log, trace, and event); an event can have any number of attributes. However, to provide semantics for such attributes, the log refers to so-called *extensions*. An extension gives semantics to particular attributes. In the context of XES, five standard extensions have been defined. These extensions are described in the so-called XESEXT XML format [13]. Here, we only mention a subset of standard attributes defined by these extensions.

- The *concept extension* defines the name attribute for traces and events. For traces, the attribute typically represents some identifier for the case. For events, the attribute typically represents the activity name. The concept extension also defines the *instance* attribute for events. This is used to distinguish different activity instances in the same trace.
- The *life-cycle* extension defines the transition attribute for events. When using the standard transactional life-cycle model, possible values of this attribute are "schedule", "start", "complete", "autoskip", etc.
- The organizational extension defines three standard attributes for events: *resource*, *role*, and *group*. The resource attribute refers to the resource that triggered or executed the event. The role and group attributes characterize the (required) capabilities of the resource and the resource's position in the organization. For example, an event executed by a sales manager may have role "manager" and group "sales department" associated to it.
- The time extension defines the *timestamp* attribute for events. Since such a timestamp is of type *xs:dateTime*, both a date and time are recorded.

Users and organizations can add new extensions and share these with others. For example, general extensions referring to costs, risks, context, etc. can be added. However, extensions may also be domain specific (e.g., healthcare, customs, or retail) or organization specific.

XES may declare particular attributes to be *mandatory*. For example, it may be stated that any trace should have a name or that any event should have a timestamp. For this purpose, a log holds two lists of *global attributes*: one for the traces and one for the events.

An XES log may define an arbitrary number of *classifiers*. Each classifier is specified by a list of attributes. Any two events that have the identical values with respect to these attributes are considered to be equal for that classifier. These attributes should be mandatory event attributes. For example, if a classifier is specified by both a name attribute and a resource attribute, then two events are mapped onto the same class if their name and resource attributes coincide.

The XES meta model shown in Figure 11 does not prescribe a concrete syntax. In principle many serializations are possible. However, to exchange XES documents, a standard XML serialization is used. Figure 12 shows a fragment of the XES XML serialization of the event log of Table 1. In the example XES log, three extensions are declared: *Concept*, *Time*, and *Organizational*. For each of these extensions a shorter prefix is given. These prefixes are used in the attribute names. For example, the *Time* extension defines an attribute *timestamp*. As shown in Figure 12, this extension uses prefix time, therefore the timestamp of an event is stored using the key *time:timestamp*.

3.2 Discovery

The first type of process mining is discovery. A discovery technique takes an event log and produces a model without using any a-priori information. An example is the α-algorithm [8]. This algorithm takes an event log

```
<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="1.0" xmlns="http://www.xes-standard.org" xes.creator="Fluxicon Nitro">
     <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
     <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
     <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.xesext"/>
     <global scope="trace">
          <string key="concept:name" value="name"/>
     </global>
     <global scope="event">
          <string key="concept:name" value="name"/>
          <string key="org:resource" value="resource"/>
          <date key="time:timestamp" value="2011-09-08T16:17:41.056+02:00"/>
     </global>
     <classifier name="Activity" keys="Activity"/>
     <classifier name="activity classifier" keys="Activity"/>
     <trace>
          <string key="concept:name" value="StevenRothery"/>
          <event>
               <string key="concept:name" value="Receive quotation"/>
               <string key="org:resource" value="John"/>
               <date key="time:timestamp" value="2010-09-16T09:28:00.000+02:00"/>
               <string key="Activity" value="Receive quotation"/>
          </event>
          <event>
               <string key="concept:name" value="Order goods"/>
               <string key="org:resource" value="Claire"/>
               <date key="time:timestamp" value="2010-09-17T15:15:00.000+02:00"/>
               <string key="Activity" value="Order goods"/>
          </event>
          <event>
               <string key="concept:name" value="Receive goods"/>
               <string key="org:resource" value="John"/>
               <date key="time:timestamp" value="2010-09-18T13:45:00.000+02:00"/>
               <string key="Activity" value="Receive goods"/>
          </event>
          <event>
               <string key="concept:name" value="Create invoice"/>
               <string key="org:resource" value="Carol"/>
               <date key="time:timestamp" value="2010-09-18T14:03:00.000+02:00"/>
               <string key="Activity" value="Create invoice"/>
          </event>
          <event>
               <string key="concept:name" value="Send goods"/>
               <string key="org:resource" value="Claire"/>
               <date key="time:timestamp" value="2010-09-21T09:05:00.000+02:00"/>
               <string key="Activity" value="Send goods"/>
          </event>
          <event>
               <string key="concept:name" value="Send invoice"/>
               <string key="org:resource" value="Carol"/>
               <date key="time:timestamp" value="2010-09-23T09:42:00.000+02:00"/>
               <string key="Activity" value="Send invoice"/>
          </event>
     </trace>
     <trace>
          <string key="concept:name" value="PeterTrewavas"/>
          <string key="creator" value="Fluxicon Nitro"/>
          <event>
               <string key="concept:name" value="Receive quotation"/>
               <string key="org:resource" value="Mike"/>
               <date key="time:timestamp" value="2010-09-14T11:38:00.000+02:00"/>
               <string key="Activity" value="Receive quotation"/>
          </event>
          ...
     </trace>
     ...
</log>
```

Figure 12: Fragment of an XES file

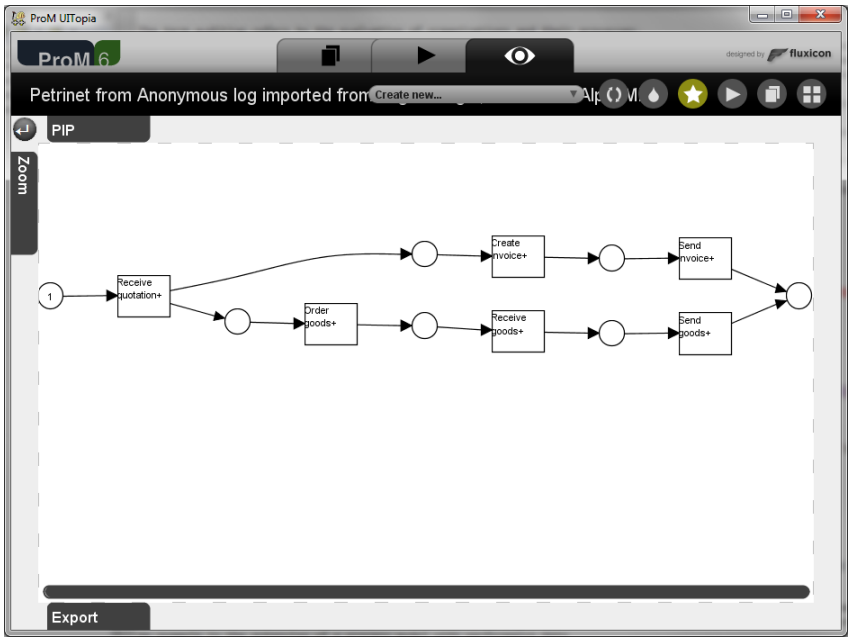

Figure 13: Process model discovered from Table 1

and produces a Petri net explaining the behavior recorded in the log. For example, given the event log as shown in Table 1, the α-algorithm is able to automatically construct a Petri net without using any additional knowledge. This Petri net is shown in Figure 13.

Many different algorithms focus on the process perspective, like the genetic miner [9] and the *Fuzzy miner* [14]. Each of the algorithms focuses on different aspects of process models, and uses different techniques. Nevertheless, algorithms that take other aspects into account also exist.

For example, if the event log contains information about resources, one can also discover resource-related models, e.g., a social network showing how people work together in an organization. Figure 14 shows the hand-over-of-work between the six resources available, where the size of the nodes corresponds to the amount of work done by the resource and the height/width ratio to the number of incoming/outgoing arcs. According to this figure, there are two clusters of people (Carol, Claire and John on the one hand and Mike, Pete, and Sue on the other) who handover work to each other.

Please note that the model obtained by discovery depends heavily on the event log at hand. As mentioned earlier, data extraction is an important step in process mining, and the question which event log is to be extracted from the execution data is key. For example, we would have been interested in how the people actually work on cases (which tasks do they typically execute in which order), we could have selected the resource as case.

3.3 Conformance

The second type of process mining is *conformance*. Here, an existing model is compared with an event log of the system [27]. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa. For instance, there may be a process model indicating that purchase orders of more than one million Euro require two checks. Analysis of the event log will show whether this rule is followed or not. Another example is the checking of the so-called “four-eyes” principle stating that particular activities should not be executed by one and the same person. By scanning the event log using a model specifying these requirements, one can discover potential cases of fraud. Hence, conformance checking may be used to detect, locate and explain deviations, and to measure the severity of these deviations.

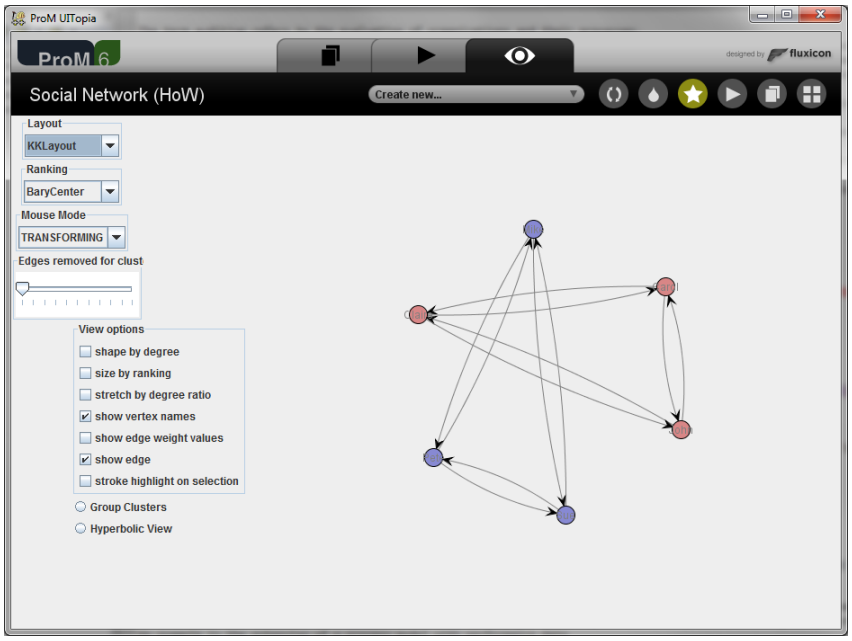

Figure 14: Social network model discovered from Table 1

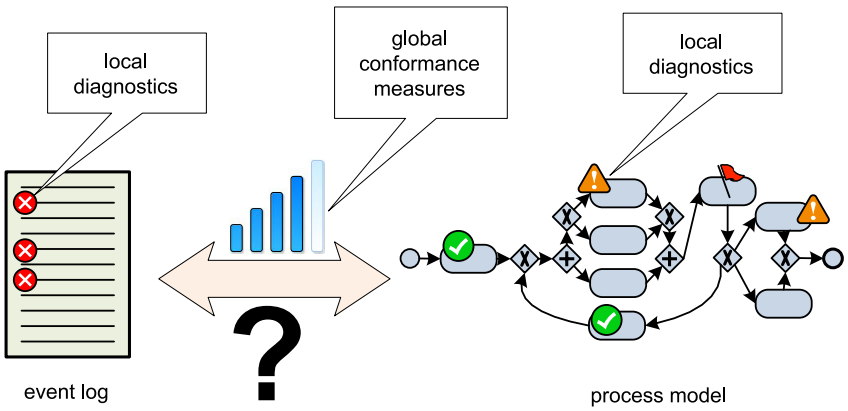

Figure 15: Conformance checking compares observed behavior with modeled behavior

When checking conformance, it is important to view deviations from two angles: (a) the model is "wrong" and does not reflect reality ("How to improve the model?"), and (b) cases deviate from the model and corrective actions are needed ("How to improve control to enforce a better conformance?"). Conformance checking techniques should support both viewpoints. Therefore, Figure 15 shows deviations on both sides.

Conformance checking is related to corporate governance, risk, compliance, and legislation such as the Sarbanes-Oxley Act (SOX) and the Basel II Accord. Corporate accounting scandals have triggered a series of new regulations. Although country-specific, there is a large degree of commonality between Sarbanes-Oxley (US), Basel II/III (EU), J-SOX (Japan), C-SOX (Canada), 8th EU Directive (EUROSOX), BilMoG (Germany), MiFID (EU), Law 262/05 (Italy), Code Lippens (Belgium), Code Tabaksblat (Netherlands), and others. These regulations require companies to identify the financial and operational risks inherent to their business processes, and establish the appropriate controls to address them. Although the focus of these regulations is on financial aspects, they illustrate the desire to make processes transparent and auditable. The ISO 9000 family of standards is another illustration of this trend. For instance, ISO 9001:2008 requires organizations to model their operational processes. Currently, these standards do not force organizations to check conformance at the event level. For example, the real production process may be very different from the modeled production process. Nevertheless, the relation to conformance checking is evident. As auditing is key to the PoSecCo project, we briefly reflect on the relation between conformance checking and auditing.

The term auditing refers to the evaluation of organizations and their processes. Audits are performed to ascertain the validity and reliability of information about these organizations and associated processes. This is done to check whether business processes are executed within certain boundaries set by managers, governments, and other stakeholders. For instance, specific rules may be enforced by law or company policies and the auditor should check whether these rules are followed or not. Violations of these rules may indicate fraud, malpractice, risks, and inefficiencies. Traditionally, auditors can only provide reasonable assurance that business processes are executed within the given set of boundaries. They check the operating effectiveness of controls that are designed to ensure reliable processing. When these controls are not in place, or otherwise not functioning as expected, they typically only check samples of factual data, often in the "paper world".

However, today detailed information about processes is being recorded in the form of event logs, audit trails, transaction logs, databases, data warehouses, etc. Therefore, it should no longer be acceptable to only check a small set of samples off-line. Instead, all events in a business process can be evaluated and this can be done while the process is still running. The availability of log data and advanced process mining techniques enables new forms of auditing. Process mining in general, and conformance checking in particular, provide the means to do so.

As example of conformance techniques, we mention the *token replay* and the *LTL checker*. The first technique replays the provided log on the provided model, and reports where deviations occur. As mentioned before, these deviations can either be in the log or in the model. For example, if we replay the log as shown in Table 1 in the BPMN model as shown in Figure 2, we could discover that none of the traces actually requires the "Receive order" task more than once. Note that this is reflected by the discovered model as shown in Figure 13. As a result, the organization could reconsider whether the loop around this task is really necessary.

The second technique is able to check LTL properties on an event log. Example of such properties include the "four-eyes" principle as mentioned earlier. For example, for the log as shown in Table 1 we could check whether the "Receive goods" and "Send goods" tasks are always executed by different persons. Figure 16 shows the results. Apparently, the principle is not violated by any of the traces in the log.

3.4 Enhancement

The third type of process mining is *enhancement*. Here, the idea is to extend or improve an existing model using information as recorded in some event log. Whereas conformance checking measures the alignment between model and reality, this third type of process mining aims at changing or extending the a-priori model.

One type of enhancement is *repair*, i.e., modifying the model to better reflect reality. For example, if two

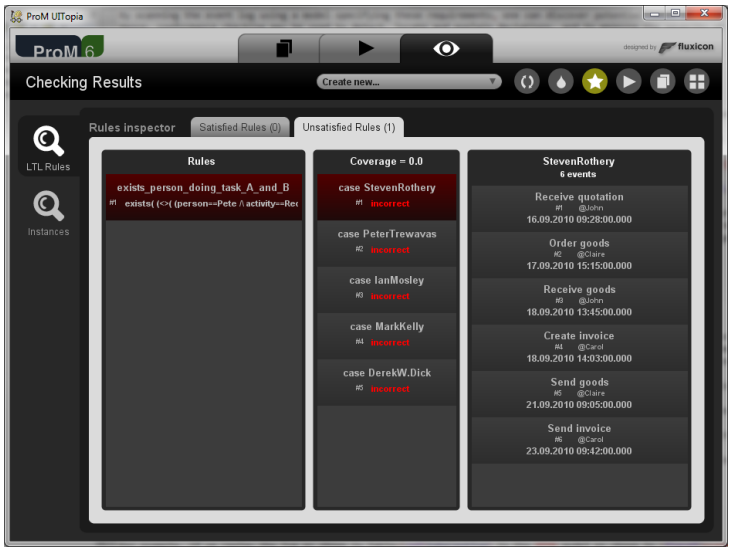

Figure 16: Results of checking the “four-eyes” principle on tasks “Receive goods” and “Send goods”

activities are modeled sequentially but in reality can happen in any order, then the model may be corrected to reflect this. Another type of enhancement is *extension*, i.e., adding a new perspective to the process model by cross-correlating it with the log. An example is the extension of a process model with performance data. For instance, by using timestamps in the event log as shown in Table 1, one can extend Figure 2 to show bottlenecks, service levels, throughput times, and frequencies. Similarly, this figure can be extended with information about resources, decision rules, quality metrics, etc.

3.5 Process Mining in PoSecCo

The main focus of PoSecCo is the policy refinement with the goal to design a golden configuration according to which the service landscape should be configured. In the enactment phase of the service landscape, it generates execution data. Process mining focuses on the analysis of this data in form of event logs. Figure 17 depicts the relationship between the BPM life cycle, process mining and PoSecCo. Based on the execution data, parts of the functional system model can be supplemented, like the resources participating in the service landscape.

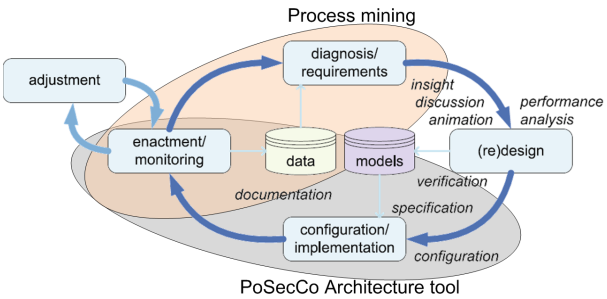

Figure 17: Position PoSecCo and process mining in the BPM life cycle

The execution data available in a service landscape gives information about the Infrastructure layer. It contains data about the resources (software components) and nodes that are running. Based on this data, parts of the Infrastructure layer of the Functional System model can be directly verified and complemented if needed. Furthermore, if sufficient execution data is available, process mining can be used to detect the interaction between services and resources.

Table 2: Message log of the *TA1* node

Message ID	Message Type	Time	Action	Resource	Content
...	...	...	...	...	...
14534123	RecSAPMsg	2010/9/08 10:25:43	Rec	SAPCon1	EDI:1243; ...
14534124	RecSAPMsg	2010/9/08 10:27:04	Rec	SAPCon1	EDI:1255; ...
14534125	TAStartJob	2010/9/08 10:27:12	Snd	W1	EDI:1255;JOB:12
14534126	TAStartJob	2010/9/08 10:27:23	Snd	W1	EDI:1243;JOB:12
14534127	TAJobSuccess	2010/9/08 10:28:42	Rec	W1	EDI:1255;JOB:12
14534128	TAJobSuccess	2010/9/08 10:29:21	Rec	W1	EDI:1243;JOB:12
14534129	RecSAPMsg	2010/9/08 10:29:25	Rec	SAPCon1	EDI:1323; ...
14534130	TAStartJob	2010/9/08 10:30:11	Snd	W1	EDI:1255;JOB:14
14534131	TAStartJob	2010/9/08 10:32:15	Snd	W1	EDI:1323;JOB:12
14534132	TAJobSuccess	2010/9/08 10:32:34	Rec	W1	EDI:1255;JOB:14
14534133	TAStartJob	2010/9/08 10:35:46	Snd	W1	EDI:1243;JOB:14
14534137	TAJobSuccess	2010/9/08 10:36:15	Rec	W1	EDI:1323;JOB:12
14534134	SendSOAPSign	2010/9/08 10:36:23	Snd	http://sign.notary.example.com	<soap:Envelope>...<m:EDI>1323</m:EDI>...
14534135	TAJobSuccess	2010/9/08 10:36:35	Rec	W1	EDI:1243;JOB:14
14534136	TASndMessage	2010/9/08 10:36:37	Snd	SSMTP1	EDI:1243; ...
14534138	TAMessageSend	2010/9/08 10:37:04	Rec	SSMTP1	EDI:1243;...
14534139	TASndMessage	2010/9/08 10:37:14	Snd	SSMTP1	EDI:1255;...
14534140	TAMessageSend	2010/9/08 10:38:12	Rec	SSMTP1	EDI:1255;...
14534141	RecSOAPSign	2010/9/08 10:45:12	Rec	http://sign.notary.example.com	<soap:Envelope>...<m:EDI>1323</m:EDI>...
14534142	TAStartJob	2010/9/08 10:45:14	Snd	W1	EDI:1323;JOB:14
14534143	TAJobSuccess	2010/9/08 10:46:16	Rec	W1	EDI:1323;JOB:14
14534144	TASndMessage	2010/9/08 10:48:12	Snd	SSMTP1	EDI:1323;...
14534145	TAMessageSend	2010/9/08 10:49:54	Rec	SSMTP1	EDI:1323;...
...	...	...	...	...	..

Consider again the running example. In the service landscape, the transaction engine records the messages it sends and receives. The information is stored in a message log as depicted in Table 2. In the content of each message, an `EDI message id` can be found. Hence, we can construct an event log in which the `EDI message id` is the case identifier, and each sending or receiving of a message becomes an event. The name of the event can be based upon parts of the content (e.g., JOB:XX) and the resource. This results in an event log as depicted in Figure 18.

The event log constructed from the message log serves as input for the analysis using ProM. First, we perform an analysis to conclude what resources are in the log, and which messages they send. This results in the diagram depicted in Figure 19. Based on this diagram, we can add the resource instances to our functional system model. Additionally, we can add each message as a data model.

Next, we want to discover the message flow of the messages being sent and received by the different resources available in the message log. Several process discovery algorithms can be used for this purpose. For example, the heuristic miner [32] results in a process model as depicted in Figure 20. Based on this process model we can identify the order in which messages were sent.

To discover which resources communicate, and in which order, we can use the performance sequence diagram analysis [16], as depicted in Figure 21. Based on this analysis, we can identify two patterns: one in which the Sign Service is being used, and one without. Based on this diagram, we can enrich the PoSecCo Functional System model with the different interactions between the different IT resources.

Last, we consider how the resources cooperate, i.e., how the resources hand over work to each other. Figure 22 depicts the social network [7] formed by the different resources. From this diagram, we can conclude that resource SAPCon1 is never called by the resources, but initiates the cooperation. Similarly, we can conclude that SSMTP1 only receives work, and never initiates new work.

In this section, we showed how current process mining techniques can be used to enrich the functional system model. In the next section, we will focus on how conformance testing can be used in PoSecCo.

```
<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="1.0" xmlns="http://www.xes-standard.org" xes.creator="ProM">
     <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
     <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
     <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.xesext"/>
     <global scope="event">
          <string key="concept:name" value="name"/>
          <string key="org:resource" value="resource"/>
          <date key="time:timestamp" value="2011-09-08T16:17:41.056+02:00"/>
     </global>
     <classifier name="Activity" keys="concept:name"/>
     ...
     <trace>
          <string key="concept:name" value="EDI-1323" />
          <event>
               <string key="concept:name" value="RecSAPMsg" />
               <string key="org:resource" value="SAPCon1" />
               <string key="sender" value="SAPCon1" />
               <string key="receiver" value="TA-1" />
               <string key="messageid" value="14.534.129" />
               <string key="content" value="EDI:1323; ..." />
               <date key="time:timestamp" value="2010-09-08T10:29:25+0200" />
          </event>
          ...
          <event>
               <string key="concept:name" value="SendSoapSign" />
               <string key="org:resource" value="http://sign.example.com" />
               <string key="receiver" value="http://sign.example.com" />
               <string key="sender" value="TA-1" />
               <string key="messageid" value="14.534.134" />
               <string key="content" value="&gt;soap:Envelope&lt;...&gt;m:EDI&lt;1323&gt;/m:EDI&lt;..." />
               <date key="time:timestamp" value="2010-09-08T10:36:23+0200" />
          </event>
          ...
     </trace>
     ...
</log>
```

Figure 18: Fraction of event log of message log in Table 2

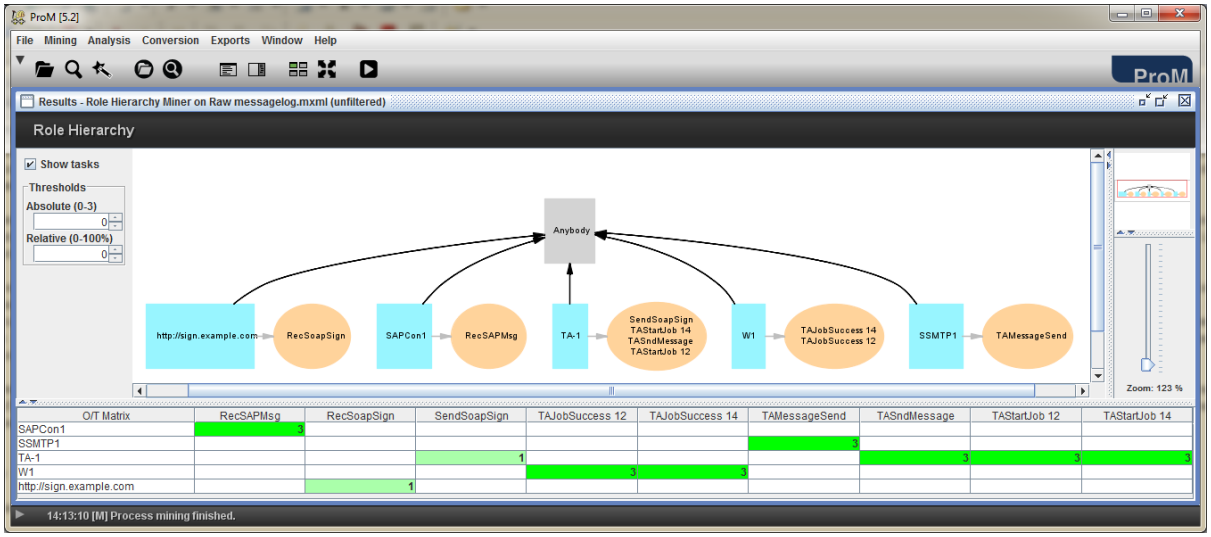

Figure 19: Resources and their messages available in the message log of Table 2

.

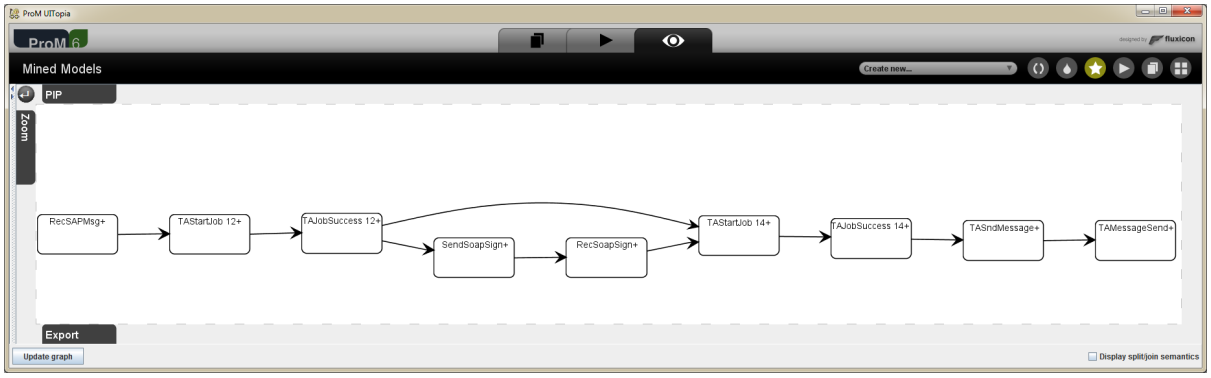

Figure 20: Process model using the heuristic miner of Table 2

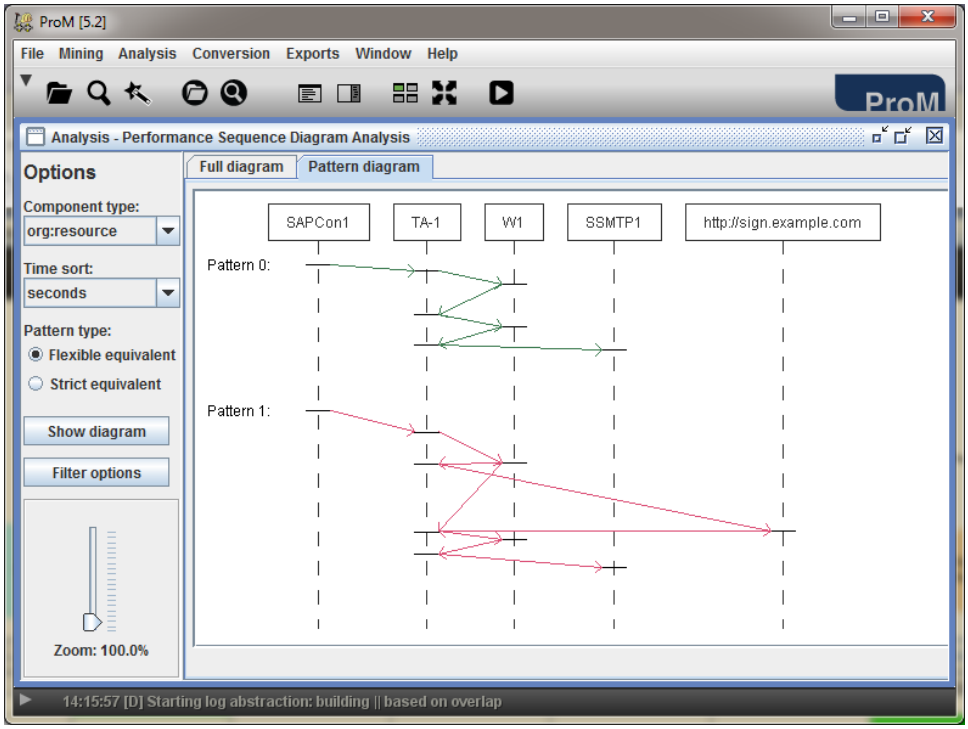

Figure 21: Communication patterns found from Table 2

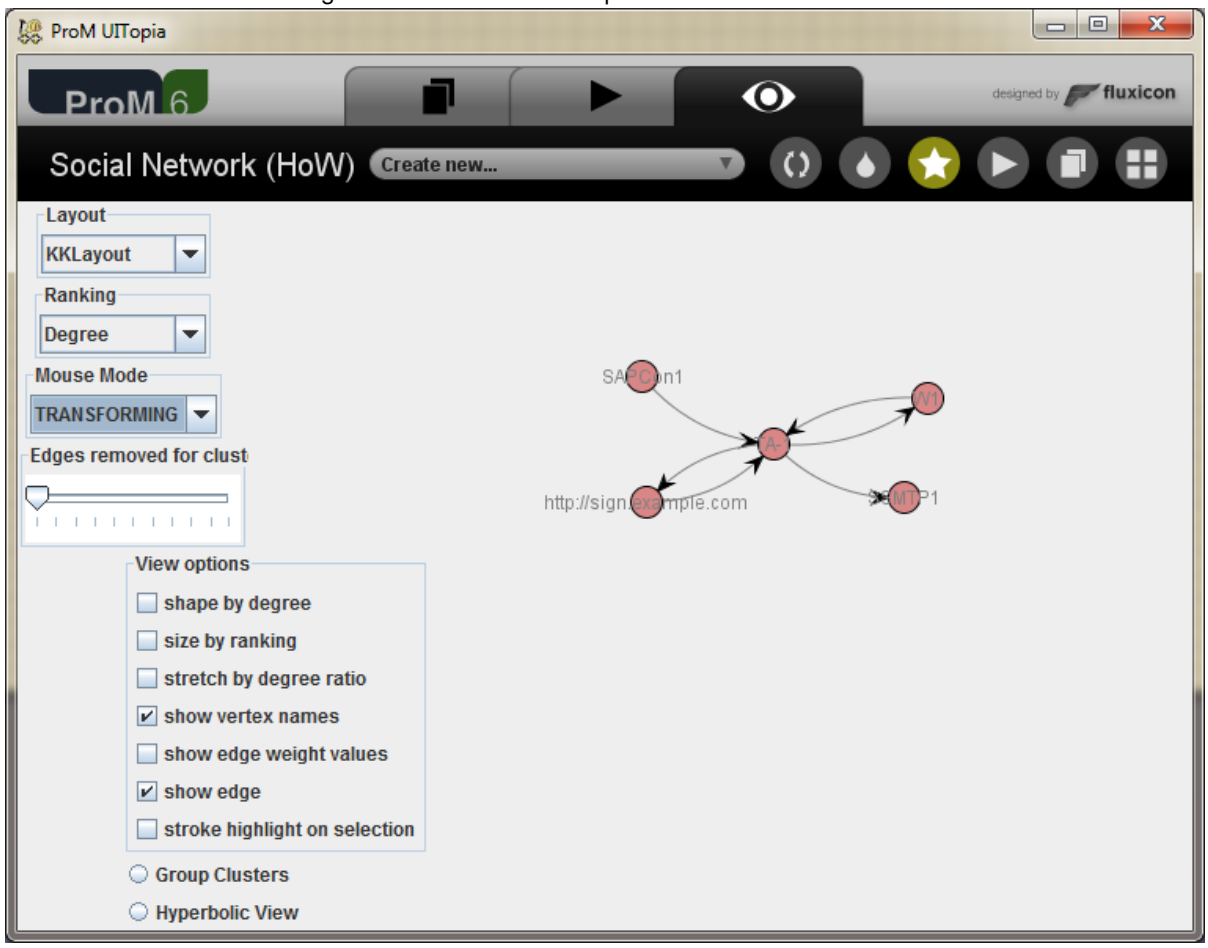

Figure 22: Handover of work of Table 2

4 SEMANTIC PROCESS MINING

A service provider offers business services to its customers. These business services are implemented by business processes. On top of these business processes, the service provider and its customers agree on the boundaries in which such a business process may be executed. Such boundaries can be *external*, like legislation as the Sarbanes Oxley Act (SOX), and legal bodies, such as BASEL II, or *internal*, like corporate rules or a code of conduct. Additionally, service level agreements (SLAs) agreed with customers define boundaries as well. Based on these requirements, a service provider implements a service landscape that executes the business processes.

In order to assure that the execution of a business process is within its boundaries, a service provider needs to constantly monitor its service landscape. The activity of checking whether the business execution adheres to the defined boundaries is called *auditing*. Traditionally, an audit can only provide reasonable assurance that a business process is compliant, i.e., that a business process is executed within the given boundaries. Auditors assess the operating effectiveness of process controls, and when these controls are not in place or functioning as expected, they typically check samples of factual data.

The omnipresence of execution data, coupled with process mining techniques enable a new form of auditing: *continuous auditing* [5], in which the execution data is used to monitor and detect compliance violations. Continuous auditing implies the automation of the auditing process. It puts high requirements on the execution data, not only the availability of execution data, but also the right data need to be available for conducting the audit. Hence, continuous auditing also requires a systematic approach to collect reliable and trustworthy execution data. This activity is known as *Business provenance*. As a consequence, business provenance is mainly an activity on the infrastructure layer of the service landscape. It requires a service provider to inspect and define at design time which resources and nodes in its infrastructure need to record which execution data.

A second aspect of automating the auditing process is the formalization of the boundaries of a business process. In order to be able to automatically verify whether the execution of a business process adheres to the defined boundaries, these boundaries should be formalized. These boundaries are formalized in *compliance rules* [23]. However, formalizing boundaries is not sufficient. Given a service landscape, the business processes and the compliance rules reside in the business layer, whereas the execution data resides on the infrastructure level. Hence, to be able to check compliance, we need to link the execution data on the infrastructure to the concepts available in the business layer. The functional system model as described in Deliverable D4.2 offers these links.

The functional system model links low level actions implemented in IT interfaces to activities on the business layer. In this way, it is possible to translate the compliance rules into low level specifications in terms of the infrastructure. However, according to the functional system model, multiple actions may be linked to many different activities, and vice versa. As a consequence, an action a of some interface i may be linked to business activities A, B and C, whereas another action b of some other interface j may be linked to business activities B,C and D. Hence, if we find in some event log the sequence $\langle a, b\rangle$, it can correspond to many different sequences of business activities: $\langle A, B\rangle$, $\langle A, C\rangle$, $\langle A, D\rangle$, $\langle B, B\rangle$, $\langle B, C\rangle$, $\langle B, D\rangle$, $\langle C, B\rangle$, $\langle C, C\rangle$, and $\langle C, D\rangle$. Hence, we need to add semantics to the event logs which enables us to link e.g. the sequence $\langle a, b\rangle$ in the event log to the sequence $\langle A, C\rangle$ of business activities.

4.1 Enriching Logs with Semantics

In semantic process mining, we use additional sources of information to enrich and analyze the event logs, as shown in Figure 23. Adding semantics to an event log can be done in many different ways. For example, each resource that raised an event, can be annotated by the department she is working for, or her role within the organization, as currently being stored using the organizational extension (Section 3.1). Or, the activity can be annotated with information about the subprocess it belongs to. However, this kind of information is typically not available in the execution data itself.

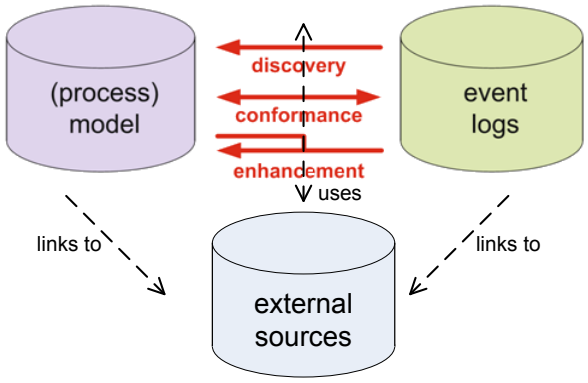

Figure 23: The use of external sources in process mining

Instead, other sources of information exist, like the information systems at the HR department, or a high-level architecture of the software system. For checking compliance, all these sources must be connected, as these additional sources contain data essential to check compliance rules. As shown in [4], a specific data model, depicted in Figure 24, together with first order logic can be used to express all kinds of compliance rules. These rules are first expressed using a fixed vocabulary, which is then translated into SQL queries on this data model. Rules like the four eyes principle, i.e., "two tasks in the same process instance should be performed by different agents", and the rule "task X can only be performed by some manager" can easily be expressed in first order predicate logic, and then translated into an SQL query. If such a query yields an empty result, the rule holds. Preliminary results [19] show that the approach works in practice. On the other hand, having a fixed vocabulary limits the flexibility to add new sources of information.

Knowledge-based techniques, and more recently, semantic web techniques like the Rich Description Format (RDF) [30] and the Ontology Web Language (OWL) [31] are designed to link sources in an open world, and allow to define and infer all kinds of rules and queries. An important difference between the semantic web paradigm and classical knowledge-based paradigm, is their view on the world. The former has an open-world assumption, meaning that if a property is missing, it is assumed to be unknown, whereas the latter has a closed-world assumption: if a property is not known to be true, it is false. The open world assumption of the semantic web allows to connect many different sources at run time, meaning that it is always possible to add new sources of information during the process of compliance checking.

As an example, consider a compliance rule that limits node access to highly qualified engineers only. The execution data only stores which user accessed a node and which actions this user performed. One solution is to store explicitly the role of the user in the log, as done in the organizational extension. Another option would be to translate the rule to the same low level as the execution data by listing all user names that represent highly qualified engineers, and state that only these users are allowed to log on. Each time this rule needs to be checked, the set of user names of allowed engineers should be retrieved, and the rules updated accordingly. Therefore, we propose a novel way to store and link data sources using the semantic web paradigm.

4.1.1 Transform Event Logs into a Semantic Model

In order to link the elements in an event log to elements in other sources of information, we translate event logs into a knowledge model, more specifically, an ontology. We base ourselves on the data model as proposed by the XES standard.

In the XES standard, we identify six main classes, which we transform into concepts in the knowledge model: *Log*, *Extension*, *Attribute*, *Trace*, *Event* and *Value*. Each of these classes becomes a concept in the ontology. As all but the Extension concept can have attached a value, we add a super concept *Attributable*. Next, we add the relations to these concepts. We identify the following relations:

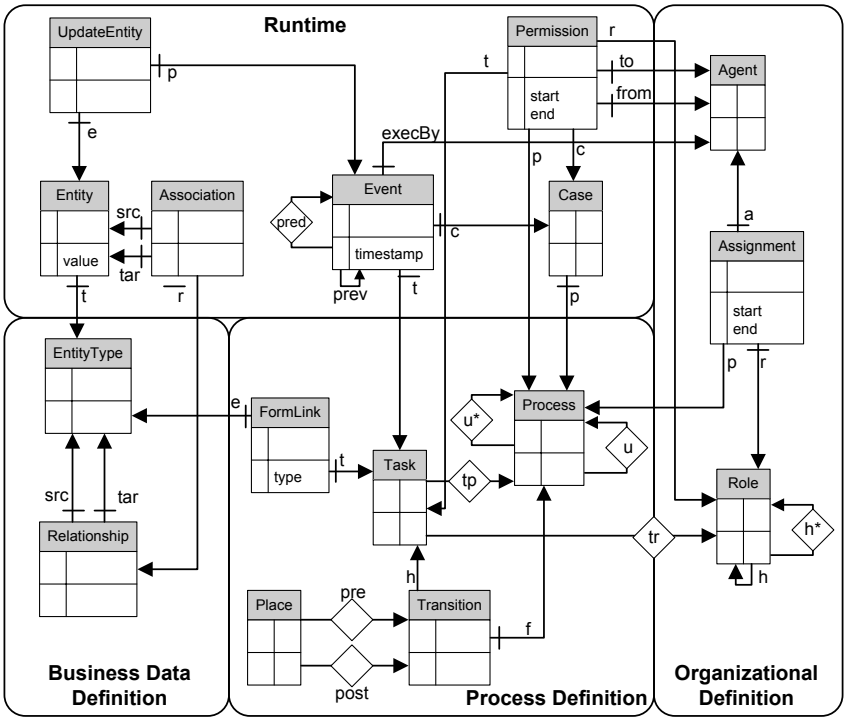

Figure 24: Initial data model for compliance checking, as proposed in [4]

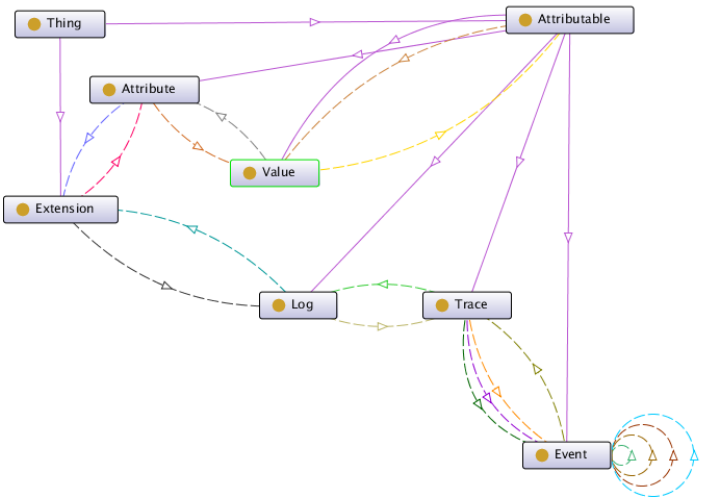

Figure 25: Ontology model of an event log

extension_of(Extension, Log) : Extension is an extension of Log.

attribute_of(Attribute, Extension) : Attribute is an attribute of Extension.

element_of(Value, Attribute) : Value is an element of Attribute.

trace_of(Trace, Log) : Trace is a trace of Log.

event_of(Event, Trace) : Event is an event of Trace.

value_of(Value, Attributable) : some Attributable (i.e., a Log, Trace, Event or Value) has value Value.

Besides, we define some derived relations. For each '_of' relation, we add the inverse 'has_' relation, like '**has_extension**(Log, Extension)' for the relation '**extension_of**(Extension, Log)'.

Note that we only store that an Attributable has some Value. The XES standard enforces that each attribute occurs at most once for each log, trace and event. Hence, a Value is uniquely related to an Attribute via the element_of relation. Hence, we do not need to store the corresponding attribute to the Attributable. As a consequence, the attribute can also be used as a relation from the attributable to its value. For example, an event e with attribute *name* from the concept extension with value A is represented in the ontology with individual e of concept *Event*, individual A of concept *Value*, and a relation *has_concept_name*(e, A). In this way, expressions become more intuitive.

For the event ordering we not only store the next event relation, but also the previous event (**prev_event**), and their transitive closures **allnext_event** and **allprev_event**, respectively. In order to traverse the events in a trace, we also store the first and last event of a trace.

next_event(Event1, Event2) : Event2 is the next event of Event1

allnext_event(Event1, Event2) : Event2 is a successor of Event1

prev_event(Event1, Event2) : Event1 is the next event of Event2

allprev_event(Event1, Event2) : Event1 is a successor of Event2

first_event(Trace, Event) : Event is the first event of Trace

last_event(Trace, Event) : Event is the last event of Trace

The resulting conceptual knowledge model is depicted in Figure 25. In the remainder of this deliverable, we refer to this ontology as the *log ontology*. Based on this model, we are able to transform any event log into an instance of the log ontology, by instantiating the appropriate concepts. Figure 26 depicts a part of an instance for some event log.

4.1.2 Semantic Extension to Event Logs

In the previous section, we have shown how to transform an event log into a knowledge model, more specifically an ontology. In order to relate individuals of the event log with other models, we need to relate the individuals of the log ontology instance with elements in other ontologies. Each instance in an ontology has a unique resource identifier (URI). Using this URI for individuals of an instantiated log ontology, we are able to relate elements between different sources. These relations can either be stored in the instantiated log ontology itself, or in the other sources. In this section, we propose an extension to the XES log format to store these relations directly in an event log. We distinguish three different ways to relate information sources. The new XES extension for adding semantics should allow to relate information in each of the three ways.

A first possibility is to directly relate individuals of the log ontology with individuals of other models, using the *object property* mechanism available in ontologies. In this way, these properties can be used in reasoning.

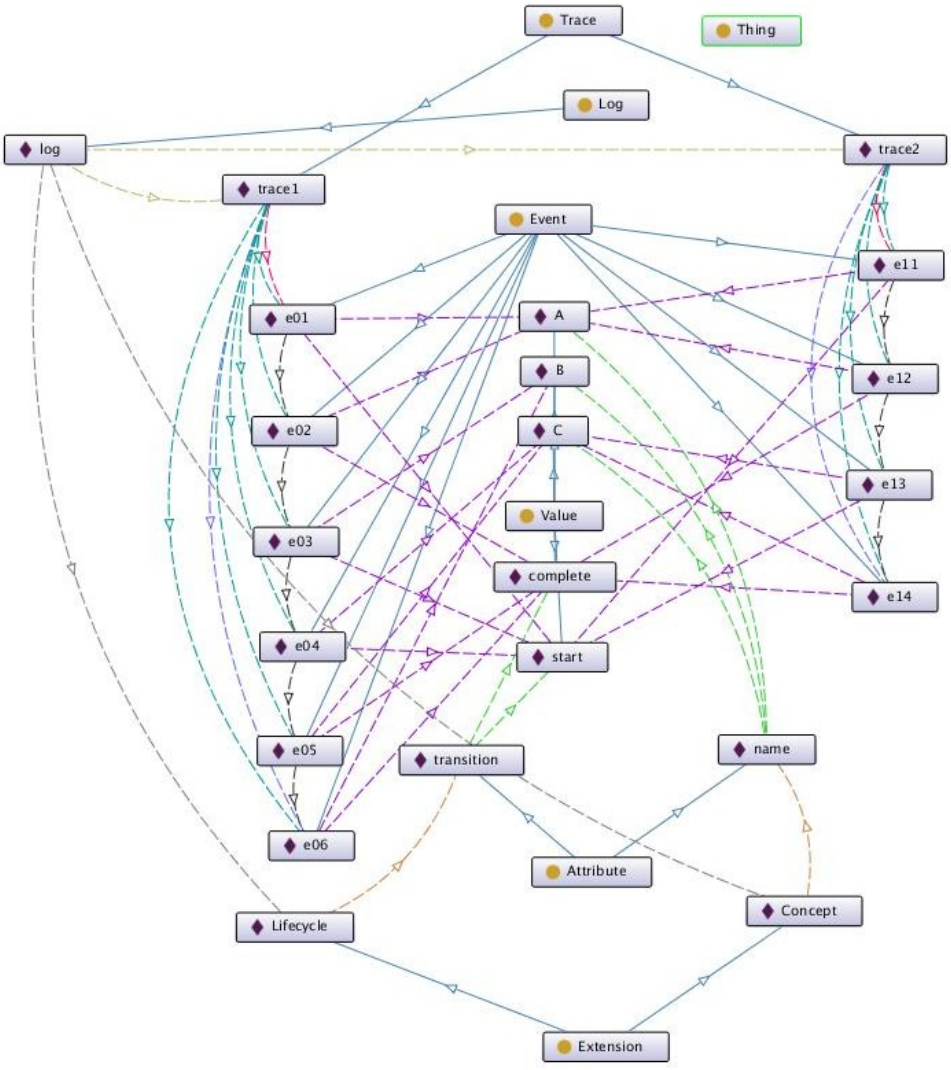

Figure 26: Instantiation of a log as an ontology

Table 3: Access log of server *TA-C* of running example

session	action	user	timestamp
...	...	...	...
SSH-1021	login	John	2010/9/08 10:30:11
SSH-1022	login	John	2010/9/08 10:39:42
SSH-1023	login	Alice	2010/9/08 10:43:11
SSH-1022	logout	John	2010/9/08 10:44:53
SSH-1021	logout	John	2010/9/08 10:45:12
SSH-1023	logout	Alice	2010/9/08 13:13:43
SSH-1024	login	Bob	2010/12/01 05:45:14
SSH-1024	logout	Bob	2010/12/04 18:34:12
SSH-1025	login	Charley	2010/12/07 13:25:12
TTY-1254	login	root	2010/12/14 13:11:12
TTY-1254	logout	root	2010/12/14 13:35:56
...	...	...	...

Secondly, the open world architecture of ontologies allows to distribute knowledge over different models. Rephrased, it is possible to have an individual that occurs in multiple ontologies. In this way, we are able to link individuals in the log ontology directly with individuals in other sources, by stating they are equivalent.

Thirdly, ontologies allow for multiple inheritance, i.e., an individual can be an instance of different concepts. In this way, we can relate individuals of an instantiated log ontology with concepts of other sources. For example, given a source ontology in which the concept *Manager* exists, we can express that an originator in the instantiated log ontology is not only a value, but also a manager. In this way, we can reason about elements of the event log as if it was a *Manager*.

4.2 Semantical Conformance Checking

In the previous section, we introduced a translation from event logs into ontologies, and showed how the individuals in the log ontology can be related to other sources of information. In this section, we introduce how we can use these semantic aspects in process mining.

Conformance focuses on the question whether the execution adheres to some specification. Classical conformance techniques focus on testing on the same layer of abstraction. In this section, we show how the addition of semantics help in bridging high level specifications and low level event logs.

Consider again the running example introduced in Section 2. As an example, we consider the next compliance rules:

1. No direct root access is allowed.
2. Only highly qualified engineers are allowed to access nodes of customer C.

For both rules, we need the access logs of each of the systems, i.e., the access log of *DB-C*, *TA-C*, *W-C*, *SAPConn-C* and *S-SMTPmailer-C*. As the sign service is outsourced, no access log exists. An example of such a log for a single system is displayed in Table 3. Based on the access logs of each of the systems, we can create an event log. As case identifier we choose the session identifier, together with an identifier for the server, e.g., the first case of the access log of Table 3 is “SSH-1 on TA-C”, as activity we choose the action. This results in an event log as shown in Figure 27.

Based on the constructed event log, we are able to check the compliance rules. The first rule can be checked using the existing LTL checker, by verifying the rule:

$$\Box(\ (Activity =' login') \implies (Resource \neq' root')\ )$$

```
<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="1.0" xmlns="http://www.xes-standard.org/" xes.creator="ProM">
     <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
     <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
     <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.xesext"/>
     <extension name="Semantic" prefix="semantic" uri="http://www.xes-standard.org/semantic.xesext" />
     <global scope="trace">
          <string key="server" value="server" />
          <string key="session" value="session" />
     </global>
     <global scope="event">
          <string key="concept:name" value="name"/>
          <string key="org:resource" value="resource"/>
          <date !key="time:timestamp" value="2011-09-08T16:17:41.056+02:00"/>
     </global>
     <classifier name="Activity" keys="concept:name"/>
     ...
     <trace>
          <string key="server" value="TA-C" >
               <string key="semantic:equivalent" value="http://example.com/customerC.owl#node-TA-C" />
          </string
          <string key="session" value="SSH-1021" />
          <event>
               <string key="concept:name" value="login" />
               <string key="org:resource" value="John" />
               <date key="time:timestamp" value="2010-9-8T10:30:11+0200" />
          </event
          <event>
               <string key="concept:name" value="logout" />
               <string key="org:resource" value="John" />
               <date key="time:timestamp" value="2010-9-8T10:45:12+0200" />
          </event>
     </trace>
     ...
</log>
```

Figure 27: Part of the event log for checking access control of Table 3

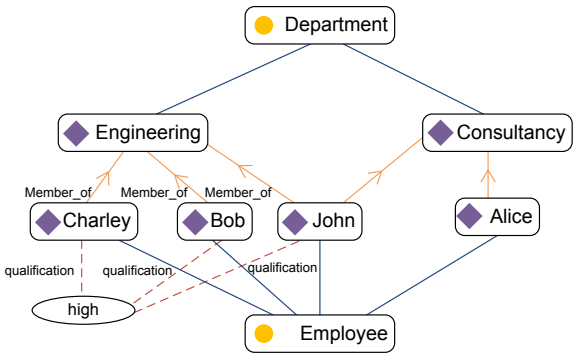

Figure 28: Ontology of user management

The LTL checker returns that instance “TTY-1254 on TA-C” violates the compliance rule.

To check the second compliance rule, we need additional information: which nodes are specific for customer C, and which users are highly qualified engineers. The information about which nodes to use can be inferred from the instance of the functional system model, which has been translated into an ontology (cf. Deliverable D3.2). In the event log, the attribute “server” is annotated with its equivalent counter part in the functional system model. In this way, we are able to express that we are only interested in traces that relate to a server that is part of the service landscape for customer C, by stating that the server of that trace runs some IT Resource that provides an IT service used by customer C, which results in the following query:

Trace **and** has_server **some** (
Node **and inverse** runs_on **some** (
ITResource **and inverse** provided_by **some** (
ITService **and inverse** uses **some** (
Institution **and** name **value** ‘C’))))

The information which users are highly qualified engineers needs to come from a user management system, like an LDAP or active directory server. An example is depicted in Figure 28. Again, we need to enrich the event log by stating which user in the event log corresponds to which user in the user management system. This can be done by adding an equivalence relation between the users of the event log and the users in the user management system. After establishing the equivalence, we are able to express the desired compliance rule, such that it returns all violating traces:

Trace **and** has_server **some** (
Node **and inverse** runs_on **some** (
ITResource **and inverse** provided_by **some** (
ITService **and inverse** uses **some** (
Institution **and** name **value** ‘C’))))
and has_event **some** (
Event **and not** (has_org_resource **some** (
Employee **and** member_of **some** (Department **and** name **value** ‘Engineering’)
and qualification **value** ‘high’)))

Although this query can be executed by any ontology reasoner, the result using the ontology of Figure 28 is not as expected. As user ‘Alice’ is not a member of the Engineering department, trace “SSH-1023 on TA-C” is a counter example of this compliance rule. The problem lies in the relation ‘member_of between the concepts ‘User’ and ‘Department’. As this relation is many-to-many, information is missing: although Alice is not member of the Engineering department, this is not stated explicitly. As a consequence, the reasoner cannot conclude that trace “SSH-1023 on TA-C” is a counter example. To resolve this, a reasoner with the closed world assumption is required [33].

Let us consider another example. Table 2 is part of a message log of the transaction engine. It contains information about which messages have been sent to which machines. In the running example of Section 2, the business process 'Sign' needs to be executed if the EDI message represents an invoice. This compliance rule can be checked by transforming the message log into an event log, as shown in Figure 18.

As it is not possible to deduct from the content of the messages whether a message is an invoice or not, we need to add additional sources of information. In this case, we need information about the kind of EDI message being processed. As a consequence, we need to process the log first to add information about the type of message, in order to check the compliance rule. Given an EDI ontology containing the different types of EDI messages, the event log can be enriched by adding for each EDI message the concept the message belongs to, as is done in the event log depicted in Figure 29.

Next, the activity 'Sign' in the business layer needs to be linked to actions in the event log. Based on the message log, we are able to relate the resource to or from which a message is sent or received to the IT Service model on the IT layer, and hence, to the business process involved. In the event log we only state for each resource to which element in the functional system model it links. In this way, we are able to formulate a query to verify the compliance rule:

```
Trace and has_EDIMessage some ( EDIInvoice )
      and has_resource some ( ITResource and providedBy some
                                        ( ITService and inverse uses some
                                                    (Instutition and name value 'C') ) )
      and not ( has_event some ( Event and has_resource some
            ( ITResource and inverse implements some
                ( ITResourceModel and exposes some
                    (ITInterfaceModel and inverse composes some
                        ( ITServiceModel and realizes some
                            (BusinessProcess and name value 'Sign') ) ) ) ) ) )
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="1.0" xmlns="http://www.xes-standard.org" xes.creator="ProM">
     <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
     <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
     <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.xesext"/>
     <extension name="Semantic" prefix="semantic" uri="http://www.xes-standard.org/semantic.xesext" />
     <global scope="event">
          <string key="concept:name" value="name"/>
          <string key="org:resource" value="resource"/>
          <date key="time:timestamp" value="2011-09-08T16:17:41.056+02:00"/>
     </global>
     <classifier name="Activity" keys="concept:name"/>
     ...
     <trace>
          <string key="concept:name" value="EDI-1323">
               <string key="semantic:elementof" value="http://example.com/edifact/#INVOIC"/>
          </string>
          <event>
               <string key="concept:name" value="RecSAPMsg" />
               <string key="org:resource" value="SAPCon1">
                    <string key="semantic:equivalent" value="http://example.com/customerC.owl#resource-SAPCon1" />
               </string>
               <string key="sender" value="SAPCon1">
                    <string key="semantic:equivalent" value="http://example.com/customerC.owl#resource-SAPCon1" />
               </string>
               <string key="receiver" value="TA-1">
                    <string key="semantic:equivalent" value="http://example.com/customerC.owl#resource-TA-1" />
               </string>
               <string key="messageid" value="14.534.129" />
               <string key="content" value="EDI:1323; ..." />
               <date key="time:timestamp" value="2010-09-08T10:29:25+0200" />
          </event>
          ...
          <event>
               <string key="concept:name" value="SendSoapSign" />
               <string key="org:resource" value="http://sign.example.com">
                    <string key="semantic:equivalent" value="http://example.com/customerC.owl#resource-SignService" />
               </string>
               <string key="receiver" value="http://sign.example.com">
                    <string key="semantic:equivalent" value="http://example.com/customerC.owl#resource-SignService" />
               </string>
               <string key="sender" value="TA-1">
                    <string key="semantic:equivalent" value="http://example.com/customerC.owl#resource-TA-1" />
               </string>
               <string key="messageid" value="14.534.134" />
               <string key="content" value="&gt;soap:Envelope&lt;...&gt;m:EDI&lt;1323&gt;/m:EDI&lt;..." />
               <date key="time:timestamp" value="2010-09-08T10:36:23+0200" />
          </event>
          ...
     </trace>
     ...
</log>
```

Figure 29: Fraction of event log of message log in Table 2, extended with semantic annotations

5 CONCLUSIONS

This deliverable discussed the use of process mining in the scope of PoSecCo. Behavioral aspects exists on all three architectural layers of a service landscape. They can be characterized as interaction of IT resources or the choreography of services and described by standard languages like BPMN or WS-CDL.

PoSecCo focuses on the design of policy and security configuration, resulting in a golden configuration of a service landscape. The service landscape is configured according to this configuration using a configuration management system. During the enactment phase of the service landscape, execution data is collected in order to be able to audit the system. The execution data may be very diverse, like the access logs of a server, or the message log of a transaction engine.

Process mining analyzes this execution data in the form of event logs. It offers the functionality to extract the behavioral information from the logs and to automatically enrich the structural landscape description. This decreases the manual effort for model population significantly. However, sufficient data is needed of a high quality. In order to use process mining within PoSecCo, a clear case and task identifier should be present in the execution data. When the data is of sufficient quality, process mining allows the PoSecCo Architecture tools to:

1. discover the different elements available on the infrastructure and IT layer, and how these elements communicate.
2. to verify whether the service landscape adheres to the configuration in the functional system model.

Whereas the enrichment of the functional system model is subject to more classic process mining techniques, semantic process mining has improved capabilities to combine the information of several event logs with other information sources. These additional information sources, together with the enriched event logs are the basis for reasoning and conformance checking.

To support the use cases described in this deliverable, we need to implement additional plugins for the open source tool ProM [3]. At least the following will be implemented during the project:

- Support to load and store ontologies;
- Support to transform event logs into ontologies;
- Support to link elements within an event log to existing elements within ontologies;
- Support for reasoning over ontologies using existing state-of-the-art reasoners, like HermiT [21], Pellet [24] and Thea [29];
- Support to verify high-level policies using ontology reasoners;

With the proper tool support implemented in ProM, semantic process mining techniques can be used to prove the conformance of the system's behavior to a given model at this given point of time. Furthermore, in combination with continuous event logging, it can deliver the conformance proof over a whole period and therefore realize an important step towards continuous auditing, as requested from customers and regulations.

REFERENCES

[1] W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin-Heidelberg, 2011.

[2] W.M.P. van der Aalst, M. Beisiegel, K.M. van Hee, D. König, and C. Stahl. An SOA-Based Architecture Framework. *International Journal of Business Process Integration and Management*, 2(2):91–101, 2007.

[3] W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, et al. ProM 4.0: Comprehensive Support for Real Process Analysis. In *Petri Nets and Other Models of Concurrency Ű ICATPN 2007*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, 2007.

[4] W.M.P. van der Aalst, K.M. van Hee, J.M.E.M. van der Werf, A. Kumar, and M.C. Verdonk. Conceptual model for on line auditing. *Decision Support Systems*, 50(3):636 – 647, 2011.

[5] W.M.P. van der Aalst, K.M. van Hee, J.M.E.M. van der Werf, and M.C. Verdonk. Auditing 2.0 using process mining to support tomorrow's auditor. *IEEE Computer*, 43(3):90 – 93, 2010.

[6] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Advanced workflow patterns. In *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, volume 1901 of *Lecture Notes in Computer Science*, pages 18 – 29. Springer-Verlag, 2000.

[7] W.M.P. van der Aalst, H. Reijers, and M. Song. Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6):549 – 593, 2005.

[8] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[9] A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.

[10] A. Alves, A. Arkin, S. Askary, et al. Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007.

[11] G. Decker, H. Overdick, and M. Weske. Oryx Ű An Open Modeling Platform for the BPM Community. In *Business Process Management*, number 5240 in Lecture Notes in Computer Science, pages 382 – 385. Springer-Verlag, 2008.

[12] IEEE Task force on Process Mining. http://www.win.tue.nl/ieeetfpm/doku.php.

[13] C.W. Günther. *XES Standard Definition*. www.xes-standard.org, November 2009.

[14] C.W. Günther and W.M.P. van der Aalst. Fuzzy mining: Adaptive process simplification based on multi-perspective metrics. In *BPM 2007*, volume 4714 of *Lecture Notes in Computer Science*, pages 328 – 343. Springer-Verlag, 2007.

[15] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, 1985.

[16] P. Hornix. Performance analysis of business processes through process mining. Master's thesis, Technische Universiteit Eindhoven, 2007.

[17] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web Services Choreography Description Language Version 1.0. `http://www.w3.org/TR/ws-cdl-10/`, November 2005.

[18] G. Keller, N. Nüttgens, and A.W. Scheer. *Semantische Process- modellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK)*. Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.

[19] J.H.W. van Loon. Design of a monitor for on-the-fly checking of business rules. Master's thesis, Technische Universiteit Eindhoven, 2011.

[20] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[21] B. Motik, R. Shearer, and I. Horrocks. A Hypertableau Calculus for SHIQ. In *Proc. of the 20th Int. Workshop on Description Logics (DL 2007)*, pages 419–426. Bozen/Bolzano University Press, 2007.

[22] Object Management Group. Business Process Modeling Notation, V2.0. http://www.omg.org/spec/BPMN/2.0/PDF/, 2008.

[23] Object Management Group. Semantics of Business Vocabulary and Business Rules (SBVR), v1.0. http://www.omg.org/spec/SBVR/1.0/PDF/, 2008.

[24] Clark & Parsia. Pellet: Owl 2 reasoner for java. http://clarkparsia.com/pellet/.

[25] A. Pnueli. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[26] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *Monographs in Theoretical Computer Science: An EATCS Series*. Springer-Verlag, 1985.

[27] A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.

[28] A.W. Scheer. *ARIS Business Process Modelling*. Springer, 1999.

[29] V. Vassiliadis, Wielemaker J., and C. Mungall. Processing OWL2 Ontologies using Thea: An Application of Logic Programming. In *OWLED*, 2009.

[30] W3C. Resource description framework (rdf): Concepts and abstract syntax, 2004.

[31] W3C. Owl 2 web ontology language, 2009.

[32] A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data Using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151 – 162, 2003.

[33] J. Wielemaker, M. Hildebrand, and J. van Ossenbruggen. Using Prolog as the fundament for applications on the semantic web. In *Proceedings of the 2nd Workshop on Applicatiions of Logic Programming and to the web, Semantic Web and Semantic Web Services*, volume 287 of *CEUR Workshop Proceedings*, pages 84–98. CEUR-WS.org, 2007.