An Ontology-Driven Process Modelling Framework

Gianluigi Greco[1], **Antonella Guzzo**[1], **Luigi Pontieri**[2], and **Domenico Saccà**[1,2]

DEIS[1], University of Calabria, Via Pietro Bucci 41C, 87036 Rende, Italy
ICAR, CNR[2], Via Pietro Bucci 41C, 87036 Rende, Italy
{ggreco,guzzo}@si.deis.unical.it, {pontieri,sacca}@icar.cnr.it

Abstract. Designing, analyzing and managing complex processes are recently become crucial issues in most application contexts, such as e-commerce, business process (re-)engineering, Web/grid computing. In this paper, we propose a framework that supports the designer in the definition and in the analysis of complex processes by means of several facilities for reusing, customizing and generalizing existent process components. To this aim we tightly integrate process models with a domain ontology and an activity ontology, so providing a sematic vision of the application context and of the processes themselves. A software architecture fully supporting our framework is also presented and discussed.

Keywords: Process Modelling, Inheritance, Workflows, Ontologies.

1 Introduction

Process modelling has been addressed for decades and a lot of frameworks have been proposed in several research fields, like Workflow Systems and Business Processes. This topic is a subject of interest in novel and attractive areas (e.g., Web/grid computing, e-commerce and e-business [4, 6]), where customization and reuse issues play a crucial role.

In this paper we devise a framework which supports designers in the definition, analysis and re-engineering of process models in complex and dynamic contexts. The main goal of our approach is to exploit the experience gained by designers over time, and somehow encoded in the process models defined so far. In order to support reuse, customization and semantic consolidation, process models are integrated in an ontological framework, which encompasses the description of the entities involved in the processes (e.g. activities and associated input/output parameters). In particular, in order to make easier the exploitation of design knowledge, we use specialization/inheritance relationships which allow us to represent similarities among process models, and can sensibly reduce the efforts for specifying a new process.

The exploitation of ontologies for describing process models and reasoning on them is not new. For example, domain and task ontologies are extensively

used by Semantic Web Services approaches (see, e.g., [2]), which are mainly devoted to automatic execution issues, rather than to exploiting design knowledge. Conversely, Business Engineering approaches (see, e.g., [10]) focus on design knowledge structuring through process taxonomies, but typically give little attention to the specification of their execution flows. Execution flows could be effectively expressed, through, e.g., one of the formalisms adopted in Workflow Management Systems ($WFMS$). Interestingly, inheritance of workflow models is investigated in [11], but principally with respect to adaptiveness and dynamic change issues, involving, e.g., the migration of executions produced by several variants a given workflow. As a consequence, the approach relies on a formal notion of inheritance, focused on behavioral features and specifically defined for workflow models represented in terms of a class of Petri nets.

By contrast, as we are mainly interested in structuring and exploiting design knowledge, in our approach the definition of specialization between process models is not necessarily bound to a rigid notion of behavioral inheritance, so leaving more freedom to the designer about the meaning of all the concepts and relationships in the knowledge base. In a sense, our framework tries to take advantage of ideas from all the above mentioned process modelling perspectives, in order to provide a complete and effective support to designers.

The formal framework for process modelling has been implemented in a prototype system that can assist the user in both design and analysis tasks, by providing a rich and integrated set of modelling, querying and reasoning facilities. In the paper, we discuss the system architecture and focus on some of its advanced functionalities, such as consistency checking and interactive ontology navigation. We devote particular emphasis to the description of a module for the automatic (re)discovering of process models. To this purpose, some *process mining* techniques [1, 5] have been recently introduced, which can derive a model for a given process based on its execution logs. Here, we extend such an approach to extracting hierarchical process models, which can be profitably integrated into our ontological framework.

2 Process Modelling Framework

In this section, we present a modelling framework, where process models can be specified in terms of ontology concepts, and then can be related among each others, to facilitate the reuse and consolidation of design knowledge.

2.1 Process Schemata

The basis for a semantic view of a process model is the ontological description of the activities and domain concepts which it involves.

Let A be a set of *activities*. An activity ontology O_A for A is a tuple $\langle \texttt{ISA}, \texttt{PARTOF} \rangle$ such that $\texttt{ISA} \subseteq A \times A$ and $\texttt{PARTOF} \subseteq 2^A \times A$, where 2^A denotes the set of all the subset of activities, such that for each $a \in A$, there exists no $A' \in 2^A$ with $a \in A'$ and A' $\texttt{PARTOF} A$. Roughly speaking, the relation a $\texttt{ISA}$ b,

for two activities a and b indicates that a is a refinement of b, while A' PARTOF a for $A' \subset A$ specifies that a consists in the execution of all the "finer" activities in A'. Hence, we say that $a \in A$ is a *complex activity* if there exists $A' \subset A$ such that A' PARTOF a; otherwise, a is said *simple*.

Running Example. In order to make clear our approach, we shall use the following example throughout the paper. Assume that a process model has to be designed to handle customers' orders in a company. The first step is to define the activities that must be carried out in the business cases. To this aim the ontology O_A includes the *Order Management* activity, which, in turn, consists of the following ones: (a) receive an order, (b) authenticate the client, (c) check the product availability, (d) ship the product, (e) send an invoice. Practically the relation PARTOF describe how a process can be broken down (or "decomposed") into sub-activities. Moreover, the relation ISA allows the designer to specialize a given activity. Some major issues related to the specialization of complex activities are discussed in the next subsection. □

Let D be the domain of our application, and let O_D be a domain ontology. The *interface* of an activity a in D is a pair$\mathcal{I}^a = \langle \texttt{InPort}^a, \texttt{OutPort}^a \rangle$ of set of concepts in D, where $\texttt{OutPort}^a$ specifies the result of the enactment of a, while $\texttt{InPort}^a$ specifies what is required for enabling a.

In general, the input concepts required by a sub-activity either are produced by other activities in the process or are (external) inputs of the process itself. Similarly, the outputs of an activity can be delivered within or outside of the process. A more detailed description of the structure of a complex activity, including the input/output dependencies between the involved sub-activities, can be obtained by the following notion of *Process Schema*.

Definition 1 (Process Schema). Let $O_{\mathcal{A}}$ be an activity ontology, O_D be a domain ontology. Let a be an activity in A. A *process schema* $\mathcal{PS}^a$ for a is a tuple $\langle I, T, a_0, F, \mathcal{CF}, \texttt{IN}, \texttt{OUT}_{min}, \texttt{OUT}_{max} \rangle$, where:

- I is the interface of a (i.e., $I = \mathcal{I}^a = \langle \texttt{InPort}^a, \texttt{OutPort}^a \rangle$);
- T is a set of activities s.t. T PARTOF a is asserted in $O_{\mathcal{A}}$;
- $a_0 \in A$ is the starting activity and $F \subseteq A$ is the set of final activities;
- $\mathcal{CF}$, referred to as *control flow graph* of $\mathcal{PS}^a$, is a relation of precedences among activities s.t. $\mathcal{CF} \subseteq (A - F) \times (A - \{a_0\})$ and $E \subseteq \mathcal{CF}^+$ is s.t.
  - $(x, y) \in E$ implies that $\texttt{InPort}^y \cap \texttt{OutPort}^x \neq \emptyset$, and
  - for each $y \in T$ and for each $c \in \texttt{InPort}^y$, either (i) $c \in \texttt{InPort}^a$ or (i) there exists $(z, y) \in E$ s.t. $c \in \texttt{OutPort}^z$
  - for each $c \in \texttt{OutPort}^a$, there exists $x \in T$ s.t. $c \in \texttt{OutPort}^x$.
- $\texttt{IN}, \texttt{OUT}_{min}$, and $\texttt{OUT}_{max}$ are three functions assigning to each activity in A^a a natural number such that (i) $\texttt{IN}(a_0) = 0$, $\forall a \in F$, (ii) $\texttt{OUT}_{min}(a) = \texttt{OUT}_{max}(a) = 0$, and (iii) $\forall x \in A^a$, $0 < \texttt{IN}(x) \leq \mathit{InDegree}(x)$ and $0 < \texttt{OUT}_{min}(a) \leq \texttt{OUT}_{max}(a) \leq \mathit{OutDegree}(a)$

where $\mathcal{CF}^+$ denotes the transitive closure of $\mathcal{CF}$, $\mathit{InDegree}(x)$ is $|\{e = (y, x) \mid e \in \mathcal{CF}\}|$ and $\mathit{OutDegree}(x)$ is $|\{e = (x, z) \mid e \in \mathcal{CF}\}|$. □

Intuitively, a process schema allow us to define the involved sub-activities, with their mutual information flow, for any activity having a significant level of complexity. For instance, the process schema for the *Order Management* activity is shown in Figure 1. *Receive Order* is the starting activity while *Send Invoice* is a final one. The values for the functions $\texttt{IN}$ and $\texttt{OUT}_{min}$ are also reported, while all $\texttt{OUT}_{max}$ values are assumed to coincide with the activity out-degree.

The informal semantics of a process schema is as follows. An activity a can start as soon as at least $\texttt{IN}(a)$ of its predecessor activities are completed. Two typical cases are: (i) if $\texttt{IN}(a) = \mathit{InDegree}(a)$ then a is an *and-join* activity, for it can be executed only after all of its predecessors are completed, and (ii) if $\texttt{IN}(a) = 1$ is called *or-join* activity, for it can be executed as soon as one predecessor is completed. As commonly assumed in the literature, we consider only *and-join* and *or-join* activities: Indeed, by means of these two elementary types of nodes, it is possible to simulate the behavior of any activity a such that $1 < \texttt{IN}(a) < \mathit{InDegree}(a)$. Once finished, an activity a activates any non-empty subset of its outgoing arcs with cardinality between $\texttt{OUT}_{min}(a)$ and $\texttt{OUT}_{max}(a)$. If $\texttt{OUT}_{max}(a) = \mathit{OutDegree}(a)$ then a is a *full fork* and if also $\texttt{OUT}_{min}(a) = \texttt{OUT}_{max}(a)$ then a is a *deterministic fork*, as it activates all of its successors. Finally, if $\texttt{OUT}_{max}(a) = 1$ then a is an *exclusive fork* (also called *XOR-fork*), for it activates exactly one of its outgoing arcs.

2.2 Process Schema Inheritance

Specialization/inheritance relationships are a mean for structuring process knowledge into different levels of abstraction. Indeed, they allow for organizing a set of related process schemata into a taxonomy, i.e. an acyclic graph where each node corresponds to a concept more general than the ones associated with its children. Undoubtedly, such a structure, for it representing similarities and differences among process models, allows to effectively exploit the design knowledge they encode.

A key point here is what is the meaning of specialization for process schemata. Diverse notions of specialization were defined in the literature, in several contexts, e.g., OO Design/Programming [9, 4], Enterprise Modelling [10, 14], and Workflow Modelling [12]. The question is particularly intriguing if one look at the behavioral feature a process schema expresses, as it implicitly represents a finite set of possible executions patterns. An interesting property which a "safe" notion of specialization should have is that all the execution instances of a pro-

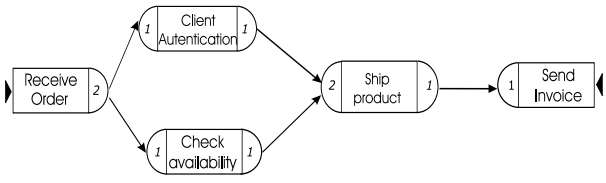

Fig. 1. Process schema for the *Order Management* activity.

cess schema are also be seen as instances of any schema which generalizes it. A behavioral notion of inheritance is adopted in [14], which concerns dataflow models, and in [3], where four variants of inheritance are introduced w.r.t. a special kind of workflow models (a class of Petri Nets, called *WF-nets*).

Notice that in our approach, we prefer not to force the designer to adopt a specific kind of specialization, so leaving her free of refining a process model in different ways. Nonetheless, our framework provides mechanisms for building (or restructuring) a process taxonomy according to the chosen notion of inheritance.

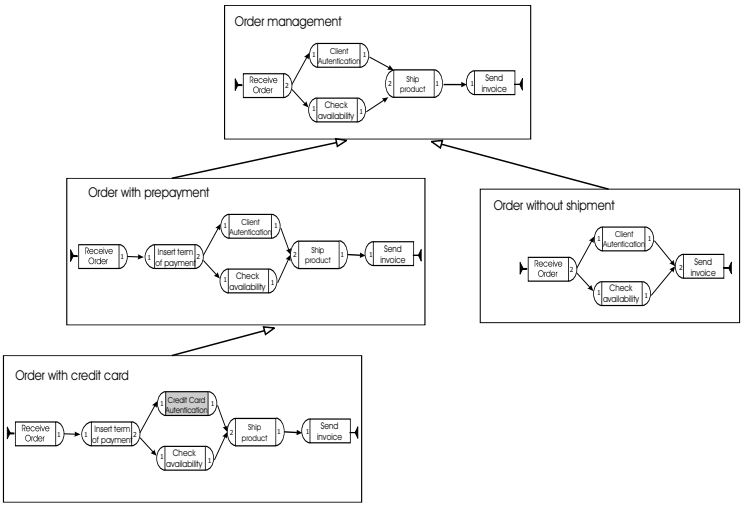

Fig. 2. An example of the inheritance in the specialization hierarchy

For example, the user could obtain derived models by specializing functional and/or behavioral features (e.g., input, output, activities, dependencies and constraints on activities) of one or more existing models. More in detail, some possible ways of specializing a given process schema are the followings:

- *Specializing an activity* involved in the original schema. Let WS be a process schema. The WS^1 is said a specialization of WS, if it is obtained from WS by replacing an activity A of WS with an activity A^1 representing a specialization of A in the ontology O_A. Note that the inverse derivation is not allowed, that is no activity of WS can be a specialization of some activity in WS^1.
- *Deleting an activity* involved in the original schema. Let WS a process schema. The WS^1 is said a specialization of WS, if it is obtained from WS by deleting an activity A of WS in WS^1. Removing an activity corresponds to exclude any process execution involving it, or, equivalently, to introduce further constraints on the process schema. Obviously, such an operation is legal only if both the initial activity and at least one of the final activities are left.

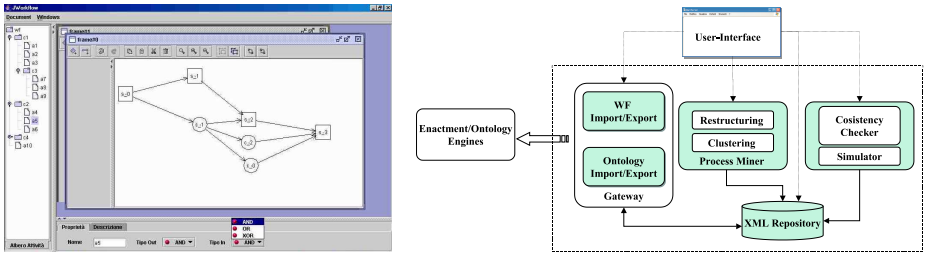

Fig. 3. System Architecture (*right*) and a screen-shot of the user interface (*left*).

- *Adding an activity* to the original schema. Let WS a process schema. Then, WS^1 is said a specialization of WS if it is obtained from WS by adding to WS an activity A^1. Such an operation corresponds to extend the original schema, more than to specialize it, since it could allow some behaviors which were not captured by that schema. Therefore, in such a case the system will warn the designer, who can make use of the restructuring facilities to recover the consistency of the schema w.r.t. some notion of behavioral inheritance.
- *Specializing the execution* either by adding further links between the activities (and/or constraints over them), or by removing links (and/or weakening some constraints). Notice that the latter operation implies again an "unsafe" specialization, which could be handled in a way similar to that discussed in the previous case.
- *Specializing the interface* of the activity associated with the original schema.

Different examples of specialization for the sample process *Order Management* are depicted in figure 2, where: the process *Order without shipment* is obtained by deleting the "ship product" activity; the process *Order with credit card* is obtained by adding the "insert term of payment" activity at the more general process; and finally, the "client authentication" activity is replaced with a more specific one ("credit card authentication") in the *Order with credit card*.

3 System Architecture

This section illustrates the main features of a software system (implemented in JAVA), intended to support the design, analysis and usage of process models. From a conceptual point of view, the architecture of the system, sketched in the right side of Figure 3, is centered upon a rich knowledge base, which stores a semantic description of the processes, according to the modelling framework presented above. Moreover, a rich set of modelling, querying and reasoning tools is provided, which allow to build and extend this knowledge base, as well as to make advantage of it in several tasks of a process model's life cycle, such as *(i)* defining or re-engineering a process model and its components, *(ii)* specializing or generalizing existing models, *(iii)* checking the workflow schema of a process and *(iv)* analyzing its behavior. The main modules in the architecture are as follows:

The **XML repository** represents the core of the system knowledge base. It is a native XML database managing the representation of process schemata and execution instances, both encoded in an XML-based format. Notably, all the semantic relationships relating schemata, activities and other domain entities are explicitly stored in the repository.

The **Ontology I/O module** offers mechanisms for connecting to, browsing and importing parts of an external ontology, provided that this exposes itself in the Web Ontology Language (OWL) [15], a semantic markup language by the World Wide Web Consortium. In addition, the module allows to make available contents of the knowledge base to the outside of the system as an ontology, still adopting the standard OWL format.

The **WF I/O module** provides the ability of translating a given process schema into an executable specification to give to a suitable enactment engine. In the current implementation of the system, Business Process Execution Language (BPEL) [8] has been chosen as such a specification language, mainly because this XML-based language represents a widely accepted notation for describing processes, fully integrated with the Web Services technology, while run-time environments supporting it are become available.

The **Consistency Checker** is in an early stage of development, and is intended to provide a number of facilities for the analysis of the defined process schemata. Currently, the module allows the user to assess the syntactic and semantic correctness of a designed process model, by providing automatic support to consistency check and schema validation analysis regarding both the static features of a model and its dynamic behavior. Nonetheless, we intend to give further support to the analysis of process behaviors, by developing a **Simulation engine** to simulate the execution of a given process in various situations. Some interesting applications of such a tool might be the investigation of the process model by means of "what if" scenarios and the comparison of alternative design choices. Details on the techniques we plan to exploit in the development of such an engine can be found in a previous work [7].

The **User Interface**, a screen-shot of which is shown on the left side of Figure 3, enables the system to be used in an easy and effective way. Notably, the whole content of the knowledge base is can be accessed by users through a general-purpose query engine associated with the *XML repository*, which is indeed based upon a native XML database. Moreover, the exploration of such data is made easier by exploiting the taxonomical structures in which the various kinds of concepts are organized, according to the specialization and partonomy relationships which relate them (look at the tree-like structure on the left side of the screen-shot).

The **Process Miner** module is substantially devoted to enable the automatic derivation of process models, conforming to our framework, based on induction techniques. Therefore, it can be of great value to the design of process models, specially when complex and heterogenous behaviors are to be modelled. It is composed of two separate components, i.e., *Restructuring* and *Clustering* mod-

ules, whose functionalities will be described in the next section, since this is the key module paving the way for an effective usage of the whole approach.

3.1 Building and Using a Process Model Knowledge Base

In this section we carefully describe the techniques implemented in the *Process Miner* module of our system architecture. These techniques can be profitably used in the re-design and analysis of process models and, hence, might help the workflow designer to model complex processes according to the formal framework we introduced in the paper. Indeed, even though workflow management systems (WfMS) are more and more utilized in enterprises, their actual impact in automatizing complex process is still limited by the difficulties encountered in the designing phase. In fact, processes have complex and often unexpected dynamics, whose modelling requires expensive and long analysis which may eventually result unviable under an economic viewpoint.

To this aim, our approach exploits a sample of process executions to build a hierarchy of process schemata conforming to our framework, which model the behaviors of the underlying process at different levels of refinement.

In order to better explain our approach, we introduce some definitions and notation. Let A_P be the set of identifiers which denote the activities involved in the process P. A *workflow trace s over A_P* is a string in A_P^*, representing a sequence of activities, while a *workflow log for P*, denoted by $\mathcal{L}_P$, is a bag of traces over A_P. Then, a set of traces produced by past enactments of the process P is examined to induce a hierarchy of process schemata representing the behavior of P at different levels of refinement.

The algorithm `ProcessDiscover`, shown in Figure 4, starts with a preliminary model $\mathcal{WS}_0^1$, which only accounts for the dependencies among the activities in P. Then it refines the current schema in iterative and incremental way, by exploiting a suitable set of features, which can discriminate different behavioral patterns.

The model $\mathcal{WS}_0^1$ is computed by mining a control flow $\mathcal{CF}_\sigma$, according to the threshold σ specifying the minimum support, through the procedure *minePrecedences*, which mainly exploits techniques already presented in the literature (see, e.g., [1, 13]). Finally, such a model is inserted as the (unique) child of $\mathcal{WS}_1^1$ and the algorithm starts partitioning this schema.

After the initialization described above, the algorithm performs two separate phases. First, it constructs the taxonomy of schemata by means of a top-down refinement of each schema, implemented by means of the recursive procedure *partition*. Each workflow schema $\mathcal{WS}_i^j$ is identified by the number i of refinements needed, and an index j for distinguishing the schemas at the same refinement level. Hence, schemas $\mathcal{WS}_{i+1}^{j'}$, for different values of j', are refinements of the schema $\mathcal{WS}_i^j$, according to the framework described in the previous section.

The procedure *partition* relies on identifying different patterns of executions by means of an algorithm for clustering the process traces $\mathcal{L}(\mathcal{WS}_i^j)$ associated to each element in the hierarchy. It is based on the projection of these traces on a suitable set of properly defined *features*. Thus, in order to reuse well know clus-

Input: A set of logs $\mathcal{L}_p$, a threshold σ, and a natural number m.
Output: A taxonomy of process models.
Method: Perform the following steps:

1 $\mathcal{CF}_\sigma(\mathcal{WS}_0^1) := minePrecedences(\mathcal{L}_p)$;
2 **let** $\mathcal{WS}_0^1$ be a schema, with $\mathcal{L}(\mathcal{WS}_0^1) = \mathcal{L}_P$;
3 $mineLocalConstraints(\mathcal{WS}_0^1)$;
4 $\mathcal{WS}_1^1 := \{\mathcal{WS}_0^1\}$;
5 $refineWorkflow(1,1)$;
6 **return** $\mathcal{WS}_1^1$;

Procedure $partition(i$: step, j: schema);

1 **if** $|\mathcal{L}(\mathcal{WS}_i^j)| > m$ **do**
2 **for each** $\mathcal{WS}_{i+1}^{j'} \in \mathcal{WS}_i^j$ **do** *//recursive processing*
3 $\mathcal{WS}_{i+1}^{j'} := partition(i+1, j')$;
4 $refineWorkflow(i+1, j')$;
5 **end for**
6 $\mathcal{WS}_i^j := \textbf{\textit{generalizeSchemas}}(\{\mathcal{WS}_{i+1}^{j'} | \mathcal{WS}_{i+1}^{j'} \in \mathcal{WS}_i^j\})$;
7 **end if**
8 **return** $\mathcal{WS}_i^j$;

Procedure $refineWorkflow(i$: step, j: schema);

1 $\mathcal{F} := \textbf{\textit{identifyRelevantFeatures}}(\mathcal{L}(\mathcal{WS}_i^j), \sigma, maxFeatures, \mathcal{CF}_\sigma)$;
2 $\mathcal{R}(\mathcal{WS}_i^j) := \textbf{\textit{project}}(\mathcal{L}(\mathcal{WS}_i^j), \mathcal{F})$;
3 $k := |\mathcal{F}|$;
4 **if** $k > 1$ **then**
5 $j' := \max\{j' \mid \mathcal{WS}_{i+1}^{j'} \in \mathcal{WS}_i^j\}$;
6 $\langle \mathcal{WS}_{i+1}^{j'+1}, ..., \mathcal{WS}_{i+1}^{j'+k} \rangle := k\text{-}means(\mathcal{R}(\mathcal{WS}_i^j))$;
7 **for each** $\mathcal{WS}_{i+1}^h$ **do**
8 $\mathcal{WS}^\vee = \mathcal{WS}^\vee \cup \{\mathcal{WS}_{i+1}^h\}$;
9 $\mathcal{CF}_\sigma(\mathcal{WS}_{i+1}^h) := minePrecedences(\mathcal{L}(\mathcal{WS}_{i+1}^h))$;
10 $mineLocalConstraints(\mathcal{WS}_{i+1}^h)$;
11 **end for**
12 **else** *//Leave of the tree*
14 **end if**;

Fig. 4. Algorithm `ProcessDiscover`

tering methods, and specifically in our implementation the *k-means* algorithm, the procedure *refineWorkflow* translates the logs $\mathcal{L}(\mathcal{WS}_i^j)$ to relational data with the procedures ***identifyRelevantFeatures*** and ***project***. Then, if more than one feature is identified, it computes the clusters $\mathcal{WS}_{i+1}^{j+1}, ..., \mathcal{WS}_{i+1}^{j+k}$, where j is the maximum index of the schemata already computed for the level $i+1$, by applying the *k-means* algorithm on the traces in $\mathcal{L}(\mathcal{WS}_i^j)$. Finally, for each schema inserted in the taxonomy the procedure *mineLocalConstraint* is applied, in order to identify local constraints as well.

The second phase of the algorithm `ProcessDiscover` is instead a bottom-up one, in which the different schemas previously computed are merged and generalized by means of the procedure ***generalizeSchemas***, which is implemented in *Restructuring* module of the system. This latter procedure, which we do not fully for lack of space, allows to produce a generalized process schema from a given set of schemata, substantially by modelling only their common features, while abstracting from details. The resulting schema will then represent a more general process (possibly abstract, i.e., not executable) than the processes modelled by the input schemata.

4 Conclusions

We have proposed a formal framework that supports the designer in the definition and in the analysis of complex and highly dynamic processes by several facilities for reusing, customizing and generalizing existent process components. The model can be used for specifying process at the desired level of details as well as to semantically characterize them by capturing both concepts of the business domain and relationships among them. The model is actually supported by means of a software architecture comprising several useful components that may be profitably exploited for designing complex processes. Among them, we have described in details the process miner module which implements an algorithm for automatically discovering an unknown process model for a given collection of log. As a further research, we plan to extend our system for also supporting advanced reasoning facilities useful for simulation purposes.

References

1. R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Proc. 6th Int. Conf. on EDBT'98*, pages 469–483, 1998.
2. Anupriya Ankolekar and Mark Burstein et al. Daml-s: Semantic markup for web services. In *Proceedings of the International Semantic Web Workshop*, 2001.
3. T.Basten and W.van der Aalst.Inheritance of behavior. *JLAP*, 47(2):47–145, 2001.
4. E. Bertino, G. Guerrini, and I. Merlo. Trigger Inheritance and Overriding in an Active Object Database System. *IEEE Trans. Knowl. Data Eng*, 12(4): 588–608,2000.
5. J.E.Cook and A.L. Wolf. Automating process discovery through event-data analysis. In *Proc. 17th Int. Conf. on ICSE'95*, pp 73–82, 1995.
6. E. Di Nitto, L. Lavazza, M. Schiavoni, E. Tracanella, and M. Trombetta Deriving executable process descriptions from UML. In *Proc. 22th Int. Conf. on ICSE'02*, pp 155–165, 2002.
7. G. Greco, A. Guzzo, and D. Saccá. Reasoning on Workflow Executions. In *Proc. 7th Int. Conf. on ADBIS03*, pp 205–219, 2003.
8. IBM. Business process execution language for web services- bpel4ws. *http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.*
9. P. kueng, P. Kawalek, and P. Bichler. How to compose an object-oriented business process model, available by http://www.cs.man.ac.uk/ipg
10. Thomas W. Malone, Kevin Crowston, and Brian Pentland et al. Tools for inventing organizations: Toward a handbook of organizational processes. *Theoretical Computer Science*, 45(3):425–443, 1999.
11. W. M. P. van der Aalst. Inheritance of business processes: A journey visiting four notorious problems. In *Petri Net Technology for Communication-Based Systems*, 383–408, 2003.
12. W. M. P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1–2):125–203, 2002.
13. W.M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *DKE*, 47(2):237–267, 2003.
14. G.M. Wyner. Defining specialization for process models. *available by http://citeseer.ist.psu.edu/467393.html*
15. W3C. OWL web ontology language reference.