UNIVERSITÀ DELLA CALABRIA
Dipartimento di ELETTRONICA, INFORMATICA E SISTEMISTICA

UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,
Informatica e Sistemistica

Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica

XIX ciclo

Tesi di Dottorato

Ontology-Driven Modelling and Analyzing of Business Process

Andrea Gualtieri

UNIVERSITÀ DELLA CALABRIA

Dipartimento di Elettronica,
Informatica e Sistemistica

Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica

XIX ciclo

Tesi di Dottorato

Ontology-Driven Modelling and analyzing of Business Process

Andrea Gualtieri

Coordinatore
Prof. Domenico Talia

Supervisore
Prof. Domenico Saccà

DEIS – DIPARTIMENTO DI ELETTRONICA INFORMATICA E SISTEMISTICA
Settore scientific disciplinare Ing-Inf/05

Everything I can do
in the One who empowers me
(Phil 4,13)

Preface

Business Process Management (BPM) is the approach to manage the execution of IT-supported business processes from a business expert's point of view rather than from a technical perspective [22]. However, currently organizations have still very incomplete knowledge of and very incomplete and delayed control over their process spaces. Enterprises have long used Workflow Management Systems (WfMSs) to describe and support the dynamic behaviour of their business. This contributes to the dominance of a simplified, workflow-centric view on business processes, i.e. business processes are reduced to the sequencing of activities. Evidence of this workflow-minded notion of processes is that languages and tools for modelling business processes focus on control flow patterns [24]. It is only recently that the weaknesses of a merely workflow-centric representation were pointed out by van der Aalst and Pesic [26]. In parallel, there has been substantial work on a more comprehensive and richer conceptual model of enterprises and their processes in the "enterprise ontology" research community, see e.g. [4], [27], [28], or recently [29]. However workflow-centric process representations and work on enterprise ontologies are still largely unconnected and so workflow-centric process representations are not very suitable for accessing the business process space at knowledge level, e.g. for the discovery of processes or process fragments that can serve a particular purpose.

This topic acquires relevance because, with the advent of Service Oriented Computing, organizations started to expose their business functionality explicitly as reusable and composable services. Business users are so oriented to reuse existing business process artifacts during process modeling, so that they are able to adapt the business processes in a more agile manner. However, as the number of business processes increases, it is difficult for them to manage the process models by themselves and to find the required business process information effectively and so they have a need for a unified view on business processes (both process models and process instances) in a machine readable form that allows querying their process spaces by logical expressions corresponding to business semantics.

Current researches [25] envision the use of Semantic technologies to increase the level of automation in BPM and to overcome the gap between the business experts and the IT people. Semantic Business Process Management (SBPM) extends the BPM approach by adopting semantic technologies. In SBPM, business process models are

based on process ontologies and make use of other ontologies, such as organizational ontology and service ontology.

The goal of SBPM is to achieve a support to users involved in the BPM lifecycle. In the literature there is no uniform view on the number of phases in the BPM lifecycle. It varies depending on the chosen granularity for identifying the phases. In [19] two roles in the lifecycle are distinguished: business analysts or business managers, who create process models and analyze process models from the business point of view, and IT engineers, who are involved in process implementation and execution phases. By this approach, four phases are considered:

– **Process Modeling:** in this phase a business analyst creates an analytical process model with by specifying the order of tasks in the business process. Business analysts have normally the possibility to specify some additional information in natural language for each element in a process model, such as what the tasks in the process are supposed to do and by whom they are expected to be performed. Process models created lack of technical information such as binding of IT services and data formats for each task and so they have a need for transforming to an executable process model.

– **Process Configuration:** In the process configuration phase a process model created in the process modeling phase is transformed and enriched by IT engineers into a process model which can be executed in a process engine [20]. The executable process model can only be partly generated from the analytical process model. The web services or the components that are needed to execute the process model have to be assigned. The same holds for data formats and data flow. The resulting executable process model can be deployed into a process engine for execution.

Process Execution: After process deployment, the process engine executes process instances by navigating through the control flow of the process model. The process engine delegates automated tasks to web services, IS components and manual tasks to human workers. In the context of SOA, the process itself should be exposed as a service and can be invoked by other processes or other clients.

– **Process Analysis:** Process analysis comprises monitoring of running process instances and process mining. Process monitoring displays information on the running process instances, such as e.g. which branches of the control flow of a running process were taken; where in the control flow the process has halted after a failure; the current variable values of a process instance, etc. Some BPMSs support also business-level monitoring, where the business analyst can specify key performance indicators of the process during process modeling, and then gets them evaluated and presented in form of dashboards during process execution. The goal of process mining is to provide information necessary for potential optimization of the process model by using process mining algorithms [21]. Process mining operates on event logs, which are produced by the process engine during process instance execution, to analyze a set of finished process instances. Process mining algorithms deduce from the event logs how the process is in reality executed. The deduced process model can then be compared with the deployed process model and thus be used for conformance checking and optimization purposes. Process mining algorithms can also be used for performance analysis of processes.

A strategic and a foundational layers define the context for these phases [30]. The strategy of an organization defines its long-term business goals, and serves as the

basis for operational decisions. The foundational layer in the SBPM lifecycle consists of ontologies which capture generic knowledge about processes, services, etc., along with domain and organization-specific knowledge.

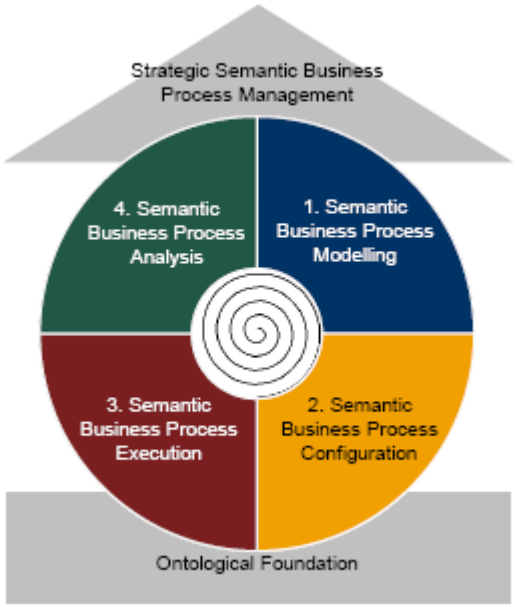

Fig. 1 Semantic BPM lifecycle

An overall approach to the SBPM should support a strategic view of all this phases, providing solutions to organize, manage, analyze and reengineer the processes running in an organization. Unfortunately the degree of automation in BPM is still unsatisfying and so many aspects of business process are still not supported by a semantic level. Moreover, a process-oriented organization is not frequently explicit inside the business context and so BPM need preliminary levels of analysis to capture the behaviour of the scenario.

Summary of contributions

The focus of this thesis is fixed on semantic business process management, aiming to offer an ontology-based support on planning, development and ex post analysis of Information System (IS) based on heterogeneous components. Starting from bases existing in literature, novel methodological and ontological instruments has been defined to represent organizational knowledge ad to support the whole BPM lifecycle, from preliminary analysis of scenario to an ex post evaluation based on the logs generated by heterogeneous functional component. Our contributes address problems that fall into the follow contexts:

Strategic BPM: a process-driven methodology for continuous information system modeling has been formulated to support the whole information system life-cycle, from planning to implementation, and from usage to re-engineering. The methodology is based on UML diagram and it is organized on three model layers, to

offer a shared view for business analyst and IT engineer both in the scenario evaluation and in the functional setting up and development.

Ontological foundation: a framework for an abstract representation of organizational knowledge has been defined to obtain a semantic model to which heterogeneous semantic context can be linked. This framework constitutes an enterprise ontology in which a functional view of organizations is annotated with respect to a conceptual view of abstract topics, allowing an approach that results more flexible than traditional ones to model a dynamic context of a real enterprise. Adopting this framework, independent operations performed by semantically heterogeneous component can be framed as steps of a unique process.

Business Process Modelling: a novel approach to model processes and their workflow is obtained by extending flow-oriented standard metamodel with respect to constructs oriented to define process taxonomies. We implement this metamodel adopting a formalism based on Disjunctive Logic Programming extended by object-oriented features to enable knowledge inference on dynamic structures of the process.

Business process configuration and execution: a general purpose solution based on a Service Oriented Architecture approach has been planned and applied to semantically integrate heterogeneous tools. By adopting an Enterprise Service Bus we obtained a physical structure able to offer: (1) on the content level, a ontology-driven conceptualization of local data, useful to obtain a synchronization on tools operating to different context; (2) on the dynamic level, a re-organization of local operation logs with respect to category of process belonging to a enterprise process ontology. A same approach allows to supply an enterprise knowledge base useful for analysis purpose.

Business process analysis: starting from the enterprise knowledge base generated by logs of execution of distributed and independent tool, new level of knowledge are extracted by adopting process mining and reasoning techniques. In the first case, process schemas have been extracted, analyzing logs with respect to characteristics of the procedures, sorts of users and temporal horizon. In the second case, hidden procedural knowledge has been discovered and taxonomic structures for the execution pattern classification have been obtained, applying logic reasoning based on Disjunctive Logic Programming, to the process schemas expressed in terms of logic rules.

Thesis outline

This thesis is conceptually organized in two parts. Part I, *The basics*, provides an introduction to elements that are useful to elaborate SBPM solutions. Part II, The advances, presents our proposal to address problems previously described. The two parts are articulated around the following chapter:

Chapter 1 offers a view on knowledge management topics and a survey on elements and languages for ontology representation focusing on OntoDLP, a formalism based on Disjunctive Logic Programming extended with object-oriented features.

Chapter 2 analyzes business process representation issue, offering a state of art of existing approaches, proposing a conceptualization of relationship between process and service and describing a standard ontology for process and service.

Chapter 3 illustrates our process-driven methodology for continuous Information System modelling.

Chapter 4 describes ontology-based framework for representing enterprise functional entities with respect to the organizational knowledge.

Chapter 5 presents a solution for analyzing loosely-structured collaborative processes based on a SOA approach that supply a knowledge base for process mining algorithms.

Chapter 6 defines a novel process metamodel and describes its implementation in OntoDLP to admit reasoning and capture of dynamic knowledge hidden in process schemas.

Acknowledgments

More than a Ph.D. qualification, the value of this experience is derived from people that shared this journey with me. Prof Domenico Saccà mapped out a route for my studies and transferred to me an approach to hold the rudder during the academic sailing, as well as everyday life storms. Whatever will be the context in which I will operate, I will always proudly stress in my curriculum that he was my supervisor.

I reserve an acknowledgment to people that trusted me and accepted to deal a work together with me: first of all prof Nicola Leone and then Massimo Ruffolo, Tina Dell'Armi, Alfredo Cuzzocrea, Antonella Guzzo, Francesco Folino, Gianlugi Greco and also profs Alessandro D'Atri and Amihai Motro. Particular thanks go to Luigi Pontieri, Marco Mastratisi and Stefano Basta for their politeness.

I was not able to publish any work together with further people that proved to be precious for me: prof Domenico Talia (team leader of the project in which a great part of this thesis has been developed), Stefania Galizia (who shared with me knowledge about several topics and awe about the future), my colleague and friend Saverio Argirò (involved in the definition of logic based process metamodel) and Edoardo Vencia and Francesco Portus (involved in the implementation of the architecture for the process mining solution). Thanks to everyone for your contribution to my work and personal improvement.

A consistent part of my thesis has been carried on during my professional experience in Exeura srl, a spin off company of the University of Calabria. An acknowledgment to Exeura president, prof Sergio De Julio, who agreed with my wish of undertake this experience. Moreover, I would like to mention the people that supported my work in Exeura: my "snack pal" Ludovico Quercia, Pina Bonavita and Isabella Di Benedetto, and my Ph.D. adventure mate, Lorenzo Gallucci. Precious presences during these years were also the ICAR and DEIS employees, especially Antonio Scudiero, Giovanni Costabile and Patrizia Mancini.

Special thanks go to the following that - out of the academic context - encouraged me. First of all my mom, who let me lean on her discreet but thoughtful and enlightening presence, and then my nonna and my uncle Ettore, that are always participants of my goals. A thought full of gratitude goes to my colleagues in the editorial staff of the Quotidiano della Calabria (Cristina Vercillo, Tiziana Aceto, Simona Negrelli, Valerio Giacoia, Adriano Mollo) that allowed me to focus on and finalize this work. Another thought full of affection goes to the persons that enriched

my everyday life, by offering their support and friendship, sharing all events happened in these years and keeping the key of the strongbox in which my emotions are preserved.

Contents

List of figures

Part I

The basics

1

Basics on ontology

Summary: The traditional information systems are able to process only explicit knowledge under structured form and use heterogeneous models and techniques for representing knowledge and manipulate them. Semantic technologies allow to increase the efficiency and effectiveness of the organizational business processes. In this chapter a survey on elements and languages for ontology representation is provided. A focus is reserved on OntoDLP, a powerful formalism based on Disjunctive Logic Programming extended with object-oriented features, adopted for ontology representation.

1.1 From information system to knowledge management system

Knowledge Management (KM) can really increase the efficiency and effectiveness of the organizational business processes. KM can contribute to the creation of value and to the intangible assets and intellectual capital growth within the enterprises. Therefore efficient KM Systems (KMS) and coherent KM strategies are needed to support the organizations in managing knowledge created, stored, distributed and applied within the business process. In particular, specific methods and instruments for organizational knowledge elicitation and representation are required for KMS and KM strategies design and implementation.

Many different kinds of organizational knowledge are wide spread within enterprises under different forms and distributed in several sources (humans and systems) inside and outside the organization. The classical distinction and generally accepted classification, due to Polanyi [72], [73] and extended by Nonaka [74], [75] identifies: “tacit and implicit knowledge”, that is the knowledge resulting from personal learning processes, present within each organization in terms of its members' personal knowing; “explicit knowledge”, generally shared and publicly accessible within the organization through formal storing and processing infrastructures. Explicit knowledge can also be classified basing on the following forms:”structured” (available in database), “semi-structured” (available in intranet and internet web sites: HTML pages, XML documents, etc.) and “unstructured” (available as textual documents: project documents, procedures, white papers, templates, etc.).

The traditional information systems present two basic problems: first they are able to process only a small portion of the whole organizational knowledge (i.e.

explicit knowledge under structured form); second they use heterogeneous models and techniques for representing knowledge and manipulate them.

A KMS must be able to support the generation, discovery, capture, store, distribution and application of a wide variety of knowledge (i.e. explicit knowledge under structured, semi-structured and unstructured forms and individual and social aspects of implicit knowledge) through related knowledge-based services. Moreover, a KMS needs capability to interoperate with already existing organizational information systems. To satisfy these requirements a KMS needs knowledge representation capabilities, that can be provided by ontology languages, able to allow the specification of the different organizational knowledge forms and kinds and to carry out an abstract representation of organizational entity supporting interoperability among different systems and organizational areas.

1.2 Ontology and knowledge representation

An ontology represents a powerful conceptual enhancement to the knowledge representation. By allowing a representation of the world conceptualization, it ensure in fact the formalization and the interchange of knowledge [78]. The “ontology” topic comes from the field of philosophy that is concerned with the study of being or existence. The term had been adopted by early Artificial Intelligence (AI) researchers, who recognized the applicability of the work from mathematical logic [77] and argued that AI researchers could create new ontologies as computational models that enable certain kinds of automated reasoning [76]. An discussion about the different connotations assumed by this term is purposed by [79].

An initial definition has been purposed by Gruber [80] and then modified in [81]. According this definition, an ontology is “explicit specification of a conceptualization”, which is, in turn, "the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold among them”. While the terms specification and conceptualization have caused much debate, the essential points of this definition of ontology are:

- an ontology defines (specifies) the concepts, relationships, and other distinctions that are relevant for modeling a domain.
- the specification takes the form of the definitions of representational vocabulary (classes, relations, and so forth), which provide meanings for the vocabulary and formal constraints on its coherent use.

Ontology engineering is concerned with making representational choices that capture the relevant distinctions of a domain at the highest level of abstraction while still being as clear as possible about the meanings of terms.

According on semantic detail required it can be opportune to develop different kinds of ontology:

- Top-level ontologies describe very general concepts like space, time, matter, object, event, action, etc., which are independent of a particular problem or domain: it seems therefore reasonable, at least in theory, to have unified top-level ontologies for large communities of users.
- Domain ontologies and task ontologies describe, respectively, the vocabulary related to a generic domain (like medicine, or automobiles) or a generic task

or activity (like diagnosing or selling), by specializing the terms introduced in the top-level ontology.

- Application ontologies describe concepts depending both on a particular domain and task, which are often specializations of both the related ontologies. These concepts often correspond to roles played by domain entities while performing a certain activity, like replaceable unit or spare component.

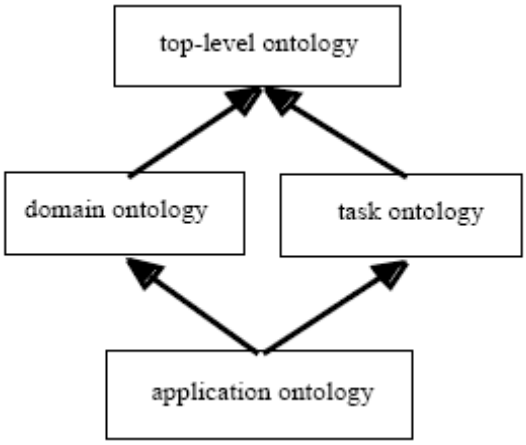

Fig. 2 Kinds of ontologies, according to semantic detail level

An ontology, which is a particular knowledge base, describing facts assumed to be always true by a community of users, in virtue of the agreed-upon meaning of the vocabulary used. A generic knowledge base, instead, may also describe facts and assertions related to a particular state of affairs or a particular epistemic state. Within a generic knowledge base, we can distinguish therefore two components: the ontology (containing state-independent information) and the "core" knowledge base (containing state-dependent information) [82].

Ontology knowledge can be specified using five components: concepts (which are usually organized by taxonomies), relations, functions, axioms, and instances.

- *Concepts* can be abstract or concrete, elementary or composite, real or fictitious; in short, a concept can be anything about which something is said, and, therefore, could also be the description of a task, function, action, strategy, reasoning process, and so on. Concepts are also known as classes, objects or categories. Instance and class attributes are commonly used in concept descriptions. The following concept attributes have been identified: instance attributes, whose value might be different for each instance of the concept; class attributes, whose value is attached to the concept, that is, its value will be the same for all instances of the concept; local attributes, same-name attributes attached to different concepts; global attributes, ones in which the domain is not specified and can be applied to any concept in the ontology.

- *Relations* are an interaction between concepts of the domain and attributes. Ontologies include binary and n-ary relations. Taxonomies are particularly kinds of binary relations used to organize ontological knowledge with respect to generalization and specialization through which simple and multiple inheritance could be applied.
- *Functions* are a special kind of relation where the value of the last argument is unique for a list of values of the n–1 preceding arguments.
- *Axioms* model sentences that are always true and can be used for several purposes, such as constraining information, verifying correctness, or deducting new information. Axioms are also known as assertions.
- *Instances* represent elements in the domain attached to a specific concept. Facts represent a relation that holds between elements, and claims represent assertions of a fact made by an instance. All these terms are used to represent elements in the domain.

1.3 Ontology specification languages

In the past years, a set of languages have been used for implementing ontologies.

One of most significant is *Ontolingua* [84], a language based on *KIF* [85] and on the Frame Ontology (FO), and it is the ontology-building language used by the Ontolingua Server. KIF (Knowledge Interchange Format) was developed to solve the problem of heterogeneity of languages for knowledge representation. It provides for the definition of objects, functions and relations. KIF has declarative semantics and it is based on first-order predicate calculus, with a prefix notation. It also provides for the representation of meta-knowledge and non-monotonic reasoning rules. The FO, built on top of KIF, is a knowledge representation ontology that allows an ontology to be specified following the paradigm of frames, providing terms such as *class*, *instance*, *subclass-of*, *instance-of*, etc. The FO does not allow to express axioms; therefore, Ontolingua allows to include KIF expressions inside of definitions based on the FO.

Another example of traditional ontology language is *FLogic* [86] - an acronym for *Frame Logic*. FLogic integrates frame-based languages and first-order predicate calculus. It accounts in a clean and declarative fashion for most of the structural aspects of object-oriented and frame-based languages, such as object identity, complex objects, inheritance, polymorphic types, query methods, encapsulation, and others. In a sense, FLogic stands in the same relationship to the object-oriented paradigm as classical predicate calculus stands to relational programming. It has a model-theoretic semantics and a sound and complete resolution-based proof theory. Applications of FLogic go from object-oriented and deductive databases to ontologies, and it can be combined with other specialized logics (HiLog, Transaction Logic), to improve the reasoning with information in the ontologies.

LOOM [87] is a high-level programming language and environment intended for use in constructing expert systems and other intelligent application programs. While the other languages are based on frame (Ontolingua) or first-order predicate calculus (KIF), LOOM adopts adescription logic approach to ontology modelling, achieving a tight integration between rule-based and frame-based paradigms. LOOM supports a

"description" language for modeling objects and relationships, and an "assertion" language for specifying constraints on concepts and relations, and to assert facts about individuals. Procedural programming is supported through pattern-directed methods, while production-based and classification-based inference capabilities support a powerful deductive reasoning (in the form of an inference engine: the classifier). Definitions written using this approach try to exploit the existence of a powerful classifier in the language, specifying concepts by using a set of restrictions on them.

Most recently, the interchange of ontologies across the web and the cooperation among heterogeneous agents placed on it is the main reason for the development of a new set of ontology specification languages, based on web standards such as *XML* [93]or *RDF* [94]and *RDF-Schema* [95].

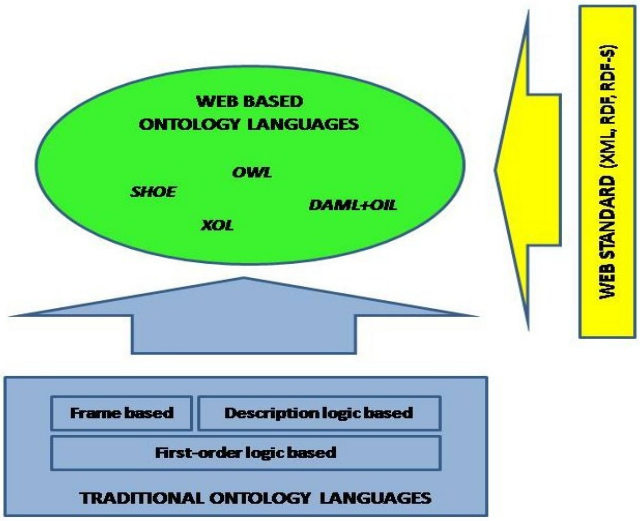

Fig. 3 Ontology languages

Many languages has been created in the context of web standards using: *XOL* [92], DARPA Agent Markup Language (*DAML*) [88], Ontology Interchange Language (*OIL*) [89], *DAML+OIL* [90], Simple HTML Ontology Extensions (*SHOE*) [91].

OWL [99] instead is the ontology language for the Semantic Web, developed by the World Wide Web Consortium (W3C) Web Ontology Working Group. OWL takes the basic fact-stating ability of RDF and the class- and property-structuring capabilities of RDF Schema and extends them. OWL can declare classes, and organise these classes in a subsumption ("subclass") hierarchy, as can RDF Schema. OWL classes can be specified as logical combinations (intersections, unions, or complements) of other classes, or as enumerations of specified objects, going beyond the capabilities of RDFS. OWL can also declare properties, organize these properties into a "subproperty" hierarchy, and provide domains and ranges for these properties, again as in RDFS. OWL can express which objects (also called "individuals") belong to which classes, and what the property values are of specific individuals. Equivalence statements can be made on classes and on properties, disjointness statements can be made on classes, and equality and inequality can be asserted

between individuals. However, the major extension over RDFS is the ability in OWL to provide restrictions on how properties behave that are local to a class. OWL can define classes where a particular property is restricted so that all the values for the property in instances of the class must belong to a certain class (or datatype); at least one value must come from a certain class (or datatype); there must be at least certain specific values; and there must be at least or at most a certain number of distinct values.

OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users:

- *OWL Lite* supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. Owl Lite also has a lower formal complexity than OWL DL.
- OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with description logics.
- OWL Full is meant for obtain maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

A framework for comparing the expressiveness and inference mechanisms of the ontology formalisms is provided by [83] in which inference mechanisms are considered to evaluate how the static structures represented in the domain knowledge can be used to carry out a reasoning process.

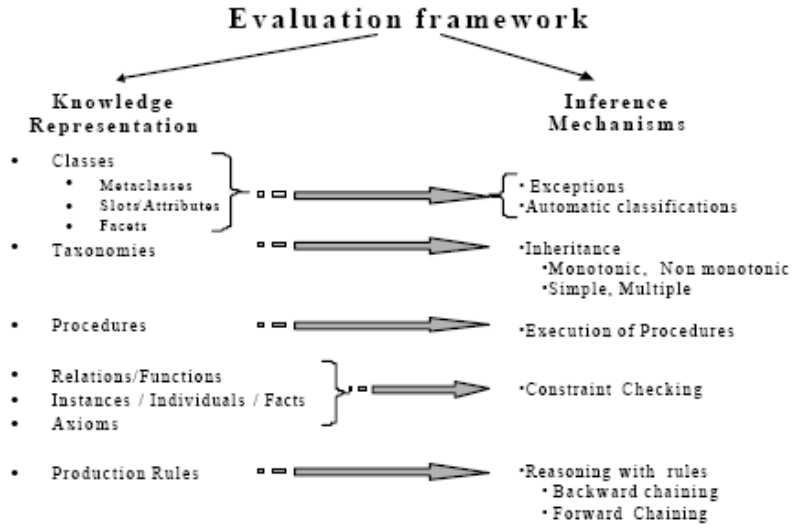

Fig. 4 An evaluation framework for knowledge languages

1.4 OntoDLP, a logic formalism for knowledge representation

OntoDLP is an extension of Disjunctive Logic Programming (DLP) by object-oriented features. In the feld of logic-based Artificial Intelligence, DLP is widely recognized as a valuable tool for knowledge representation, commonsense reasoning and incomplete-knowledge modelling [64,65,66,67,68,69,70,71]. OntoDLP combines the expressive and deductive power of DLP (capture the complexity class $\sum^2_P$) with the facilities of the object-oriented paradigm for a natural and effective real-world knowledge representation and reasoning. In particular, the language includes, besides the concept of relations, the object-oriented notions of *classes*, *objects* (class instances), *object-identity*, *complex-objects*, *(multiple) inheritance*, and the concept of modular programming by mean of *reasoning modules*. In the following, an overview of the language is given by informally describing its most significant features and by giving language use examples for the representation of some of the main concepts related to the workflow domain. Classes can be declared in OntoDLP by using the keyword *class* followed by the class name and by a comma separated list of attributes. Each attribute is a couple (*attribute-name* : *attribute-type*). The attribute-type is either a user-defined class, or a built-in class (in order to deal with concrete data types, OntoDLP makes available two built-in classes *string* and *integer*). For instance, a generic process can be represented by declaring the class process with an attribute of type string as follows:

```
class process(name:string).
```

Objects, that is class instances, are declared by asserting new facts. An instance for the class process, can be declared as follows:

```
#1:process(name:"Web sale order").
```

The string \Web sale order" values the attribute name; while #1 is the *object-identifier (oid)* of this instance (each instance is equipped by a unique oid). Classes can be organized in a taxonomy by using the *isa* relation. For example, a *common_node* is a node characterized by a possible asynchronously activation. This class specialization can be represented in OntoDLP, declaring class common node as an extension of class node with a new attribute asynchronous:

```
class common_node isa {node}
(asynchronous:string).
```

Instances of the class common node are declared as usual, by asserting new facts:

```
#2:common_node(name:"Password required", asynchronous:"true").
#3:common_node(name:"Registration required",asynchronously:"false").
```

Like in common object-oriented languages with inheritance, each instance of a sub-class becomes, automatically, an instance of all super classes (*isa* relation induces an inclusion relation between classes). In the example, “Passwor required” and “Registration required" are instances of both node and common node. Moreover, sub-classes inherit attributes from all super-classes. In the example, thecommon node class inherits attribute name of class node and declares a new local attribute named asynchronous.

The language provides a built-in most general class named *Object* that is the class of all individuals and is a superclass of all OntoDLP classes. Also multiple inheritance is supported. Attribute inheritance in OntoDLP follows the strategy adopted in the COMPLEX language, for a formal description refer to [63].

The possibility to specify user-de¯ned classes as attribute types allows for complex objects or nested classes, i.e. objects made of other objects. For example, the class transition, besides the name of type string, is characterized by two attributes of the user-defined type node.

```
class transition(name:string,from:node,to:node).
```

The following declaration of class create timer includes, besides timer name and duedate, an attribute of type action, namely, action to execute.

```
class action(name:string).
class create_timer isa {action}
(timer_name:string,duedate:integer,action_to_execute:action).
```

Note that this declaration is “recursive” (both action and create timer are of type action). An instance of class create timer can be specified as follows:

```
#4:create_timer(name:"Sell timer creation", timer_name:"Sell timer",
duedate:30,action_to_execute:#5).
```

where the oid #5 identifies a selling action:

```
#5:action(name:"Sell").
```

Instance arguments can be valued both specifying object identifiers and by using a nested class predicate (complex term) which works like a function. For example, the action to execute is specified by a complex term in the following declaration:

```
#6:create_timer(name:"Auction timer creation",timer_name:"Auction
timer",duedate:60,action_to_execute:action(name:"Auction").
```

Relations represent relationships among objects. Base relations are declared like classes and tuples are specified (as usual) asserting a set of facts (but tuples are not equipped with an oid). For instance, the base relation contains, and a tuple asserting that the process identified by oid #1 contains the common node identified by oid #2, can be declared as follows:

```
relation contains(process:process,node:node).

contains(#1,#2).
```

Classes and base relations are, from a data-base point of view, the extensional part of the OntoDLP language. Conversely, derived relation are the intensional (deductive) part of the language and are specified by using reasoning modules.

Reasoning modules, like DLP programs, are composed of logic rules and integrity constraints. OntoDLP reasoning modules allow one to exploit the full power of DLP. As an example, consider the following module, encoding the path search problem between two nodes in a process schema.

```
relation path(from:node, to:node).

module(path){

path(from:X,to:Y) :- T:transition(from:X,to:Y).

path(from:X,to:Y) :- T:transition(from:X,to,Z), path(from:Z,to:Y).}
```

The OntoDLP language is supported by OntoDLV, a system based on Answer Set Programming (ASP) for the specification and reasoning on enterprise ontologies [100]. OntoDLV supports a powerful interoperability mechanism with OWL, allowing the user to retrieve information also from OWL Ontologies and to exploit this information in OntoDLP ontologies and queries. The system is already used in a number of real world applications including agent-based systems, information extraction, and text classification applications.

2

Basics on process and service

Summary: Process models can be used for planning, simulation and automatic execution of business processes, e.g. in Workflow Management Systems and Process Brokers. In this context the process and the service representation issue are intersected. An analysis of expressiveness of existing solutions is required because of the lack of standard for process and service modelling. To obtain this goal, a conceptualization of topics is provided and correlated to existing ontological approaches existing for Semantic Business Process Managemen.

2.1 A conceptualization of process and service

A process is defined as a "a set of one or more linked subprocess or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships" [50]. An activity is a description of a piece of work that forms one logical step within a process; it may be a manual activity, which does not support computer automation, or a automated activity. In the same context, a workflow is defined as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". By the process point of view, focus is on the goal partitioning, on the sequence and synchronization of task, on the human and technological resource assignments.

A service, instead, can be considered as a set of method exposed and enriched by an interaction syntax and usage semantic description. A web service is a particular web-available, self-describing software component, that is able to execute a specific task eventually establishing an interaction with other web services.

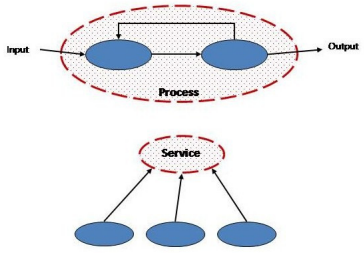

Fig. 5 Process and service

We assume that a process element (e.g. a process or a subprocess) *P* is:

P: <I, F, S,O>

where:

I is the input set to the process;

F is the function describing the process;

S is the current state of the process execution;

O is the output set of the process.

It is possible to adopt either a *structured* or an *atomic* function to describe the process. A structured function is based on a process schema to model the process execution pattern. An atomic function is either an *elementary* function or an *abstract* function: an elementary function describes a one-step process, i.e. an activity, while an abstract function offers a summarizing view on a process that should be composed by more tasks.

As a consequence we can consider a service as a composition of one or more service element S, each one is a particular kind of process defined as follow:

S: <I, F_{ab}, S_i, O>

where:

I is the input set to the service;

F_{ab} is the abstract function describing the service background;

S_i is the current internal state of the service enactment;

O is the output set of the service.

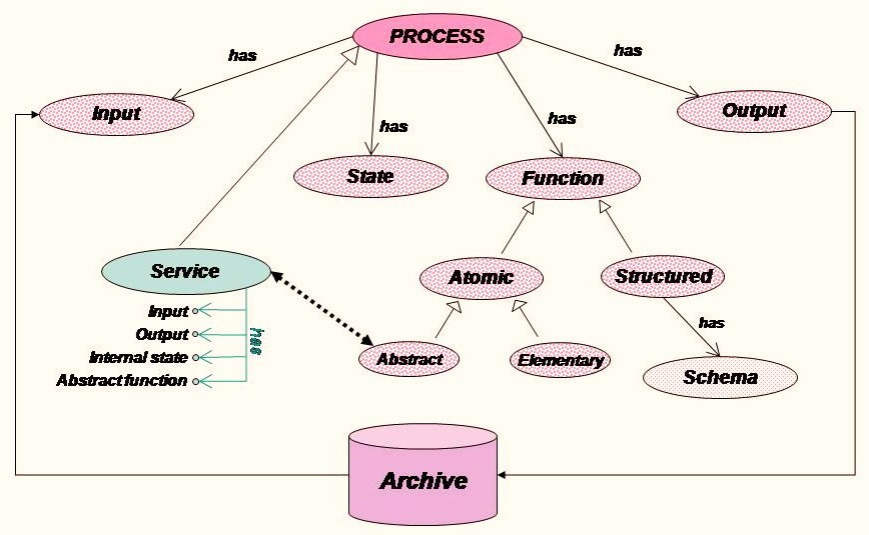

Fig. 6 Conceptualization of process and service

When a service receives a claim it enact an internal procedure to process the input and generate the expected output. A user who gains access to a service ignores what is the pattern of execution of the process that constitutes a background of the method that he invokes. The matching between exposed service (or method) and user process (or subprocess) is based on the correspondence between the input, the output and the abstract function describing a process (or a subprocess) as well as a service.

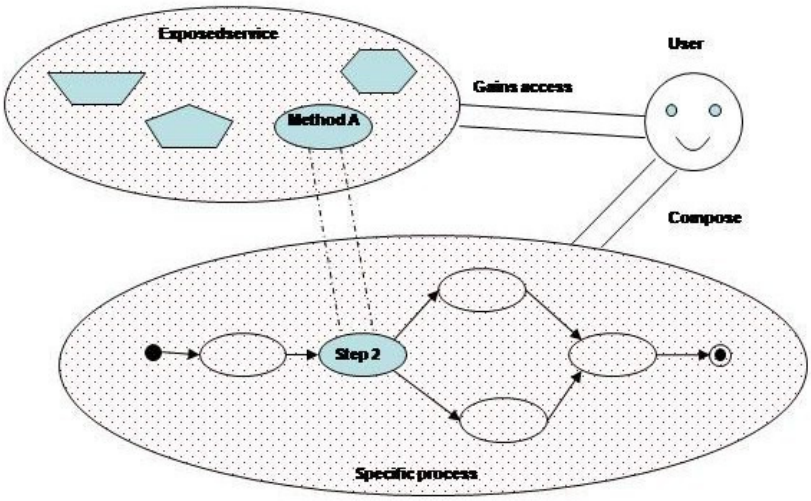

Fig. 7 Process composition using services

2.2 A three level view on process

Process management phases require both definition and analysis of process models. A large number of formalisms and approaches have been already proposed to support the design of processes [57] [58]. A cornerstone for characterizing a formalism is the specific metamodel adopted, which is an high level and platform-independent definition of the workflow items which are admitted. Many workflow systems refer an explicit metamodel, others have an implicit one, imposed by the offered features.

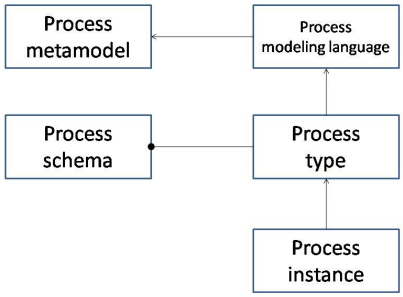

Fig. 8 Three levels of process modelling

Essential requirements in a metamodel are an unambiguous semantics and the capability to express the main elements of a workflow, e.g., according with [59]: decomposition of process in activities; definition of control-flow rules among

activities; assignment of activities to execution entities; annotation of input and output elements to each activity.

The implementation of a metamodel item is demanded to specific languages, allowing to define process schemas, each of them establishing a pattern of execution. The process schema has to be adapted to its changing environment, reflecting, e.g, new customers requirements and re-engineered business procedures. Therefore, process schema evolution, i.e., the modification of the process model over time, should be supported. Process schema evolution may include the creation of new execution pattern and the modification or deletion of existing ones.

Every time that a process execution runs, a process instance is generated, and a process log is recorded. Respect on process schema, an instance impose a value attribution to the variables defined and so in instances alternative patterns that in a process schema are connected to the analysis of local conditions are going to disappear. By analyzing a great number of instance logs it is possible to modify an existing process schema or to generate a new one [17].

2.3 Paradigms for process modelling

A standard formalism to represent processes and workflows does not exist in literature. Several works have proposed a large variety of methods and approaches to represent dynamic knowledge. Two significantly different paradigms are defined to provide a formal model for expressing executable processes [49]: one of these adopt an explicit graph representation model, the other one is oriented to a block structure analogous to programming languages. Each ones utilizes activities as the basic components of process definition. In each, activities are always part of some particular process. Each has instance-relevant data which can be referred to in routing logic and expressions. Another formalism, based on the use of Petri Nets models, provides a consistent framework to derive interesting results about structural properties of workflow [53].

2.3.1 Direct graph vs Block-structured

The first formal model, adopted by XPDL standard provided by the Workflow Management Coalition (WfMC) [50], is conceived of as a graph-structured language with additional concepts to handle blocks. Scoping issues are relevant at the package and process levels. Process definitions cannot be nested but they should be defined introducing a particular activity that is a process invoice. Routing is handled by specification of transitions between activities. The activities in a process can be thought of as the nodes of a directed graph, with the transitions being the edges. Conditions associated with the transitions determine at execution time which activity or activities should be executed next

The second formal model, adopted by the BPML standard [51] and afterwards by BPEL [52] is a block-structured programming approach, allowing recursive blocks but restricting process definitions and declarations to the top level. Within a block graph-structured flow concepts are supported to a limited extent, constrained by inheritance from previous generation workflow software (only acyclic graphs, hence

no loops; some constraints on going across block boundaries; a complicated semantics for determining whether an activity actually happens).

Translation of blocked-structured flow control (routing) into a graph structure presents no fundamental difficulties. The reverse is more problematic. This can be facilitated by imposing a set of restrictions on the graph structure that guarantee it to be "well-structured".

2.3.2 Standards oriented to the process enacting

Connected to the Business Process Execution Language (BPEL) a whole family of formalisms has been developed. In particular, BPEL extension like BPEL4WS are oriented to the web services orchestration: in this way every activity of the process can be demanded to a specific web service able to execute it. Another evolution of BPEL is BPELJ, a formalism that enables Java and BPEL to cooperate by allowing sections of Java code, called Java snippets, to be included in BPEL process definitions. Snippets are expressions or small blocks of Java code that can be used for things such as: loop conditions, branching conditions, variable initialization, web service message preparation, logic of business functions.

Also JBoss community provides a formalism to integrate process definition elements and java code. The Java Process Definition Language allows a process definition as a combination of a declaratively specified process graph and, optionally, a set of related java classes [54]. The java classes can then be made available to the jBPM runtime environment, that is the workflow engine integrated or linked by all the open source tools based on JBoss platform. Just for this reason, jBPM is one of the most adopted solution to implement workflow execution. Moreover, JPDL structure allows grouping of nodes based on the super-state construct. From this point of view, JPDL is a promising solution to capture logs of workflow execution provided by a wide number of open source tools and organize them in hierarchies based on properties of super-states.

2.4 Ontologies of process and service

Since process and service concepts are so strictly connected, an ontology-based approach to the BPM is to combine to the Semantic Web Service (SWS) technology for providing suitable knowledge representation techniques. Applying ontology to a service oriented architecture can help to identify the binding information of business process and service, increase the reusability of existing business processes and services, and accelerate the development of application. In last years, many approaches have been proposed to support the development of Semantic Web Service frameworks [61]. OWL-S purposes a view of process and service closed to the ones formalized in 2.1. WSMF introduces a level to describe business logic of services orchestration.

2.4.1 The OWL-S approach

OWL-S (previously DAML-S [55]) consists of a set of ontologies designed for describing and reasoning over service descriptions. OWL-S combines the expressivity

of description logics (in this case OWL language) and the pragmatism found in the emerging Web Services Standards, to describe services that can be expressed semantically, and yet grounded within a well defined data typing formalism. It consists of three main upper ontologies: the Profile, Process Model and Grounding.

The Profile is used to describe services for the purposes of discovery; service descriptions (and queries) are constructed from a description of functional properties (i.e. inputs, outputs, preconditions, and effects - IOPEs), and non-functional properties (human oriented properties such as service name, etc, and parameters for defining additional meta data about the service itself, such as concept type or quality of service). In addition, the profile class can be subclassed and specialized, thus supporting the creation of profile taxonomies which subsequently describe different classes of services.

OWL-S Process models describe the composition or orchestration of one or more services in terms of their constituent processes. This is used both for reasoning about possible compositions (such as validating a possible composition, determining if a model is executable given a specific context, etc) and controlling the enactment/ invocation of a service. Three process classes have been defined: the *composite*, *simple* and *atomic* process. The atomic process is a single, black-box process description with exposed IOPEs. Inputs and outputs relate to data channels, where data flows between processes. Preconditions specify facts of the world that must be asserted in order for an agent to execute a service. Effects characterize facts that become asserted given a successful execution of the service, such as the physical side-effects that the execution the service has on the physical world. Simple processes provide a means of describing service or process abstractions – such elements have no specific binding to a physical service, and thus have to be realized by an atomic process (e.g. through service discovery and dynamic binding at run-time), or expanded into a composite process. Composite processes are hierarchically defined workflows, consisting of atomic, simple and other composite processes. These process workflows are constructed using a number of different composition constructs, including: Sequence, Unordered, Choice, If-then-else, Iterate, Repeat-until, Repeat-while, Split, and Split+join. The profile and process models provide semantic frameworks whereby services can be discovered and invoked, based upon conceptual descriptions defined within Semantic Web (i.e. OWL) ontologies.

The grounding provides a pragmatic binding between this concept space and the physical data/machine/port space, thus facilitating service execution. The process model is mapped to a WSDL description of the service, through a thin grounding. Each atomic process is mapped to a WSDL operation, and the OWL-S properties used to represent inputs and outputs are grounded in terms of XML data types. Additional properties pertaining to the binding of the service are also provided (i.e. the IP address of the machine hosting the service, and the ports used to expose the service).

2.4.2 The WSMF approach

The Web Service Modelling Framework (WSMF) provides a model for describing the various aspects related to Web services. Its main goal is to fully enable e-commerce by applying Semantic Web technology to Web services. WSMF is the product of research on modelling of reusable knowledge components [56]. WSMF is based on two complementary principles: a strong de-coupling of the various

components that realize an e-commerce application; and a strong mediation service enabling Web services to communicate in a scalable manner. Mediation is applied at several levels: mediation of data structures; mediation of business logics; mediation of message exchange protocols; and mediation of dynamic service invocation. WSMF consists of four main elements: ontologies that provide the terminology used by other elements; goal repositories that define the problems that should be solved by Web services; Web services descriptions that define various aspects of a Web service; and mediators which bypass interoperability problems. WSMF implementation has been assigned to two main projects: Semantic Web enabled Web Services (SWWS) [60]; and WSMO (Web Service Modelling Ontology) [28]. SWWS will provide a description framework, a discovery framework and a mediation platform for Web Services, according to a conceptual architecture. WSMO will refine WSMF and develop a formal service ontology and language for SWS. WSMO service ontology includes definitions for goals, mediators and web services. A web service consists of a capability and an interface. The underlying representation language for WSMO is F-logic. The rationale for the choice of F-logic is that it is a full first order logic language that provides second order syntax while staying in the first order logic semantics, and has a minimal model semantics. The main characterizing feature of the WSMO architecture is that the goal, web service and ontology components are linked by four types of mediators as follows:

- OO mediators link ontologies to ontologies,
- WW mediators link web services to web services,
- WG mediators link web services to goals, and finally,
- GG mediators link goals to goals.

Since within WSMO all interoperability aspects are concentrated in mediators the provision of different classes of mediators based on the types of components connected facilitates a clean separation of the different mediation functionalities required when creating WSMO based applications.

3

A process-oriented methodology for information system design and implementation

Summary: In this chapter a novel process-driven methodology for continuous information system modelling is presented. Our approach supports the whole information system life-cycle, from planning to implementation, and from usage to re-engineering. The methodology includes two different phases. First, we produce a scenario analysis adopting a Process-to-Function approach to capture interactions among organization, information and processes; then, we produce a requirement analysis adopting a Function-for-Process and package-oriented approach. Finally, we deduce an ex-post scenario analysis by applying process mining techniques on repositories of process execution traces. The whole methodology is supported by UML diagrams organized in a Business Model, a Conceptual Model, and an Implementation Model..

3.1 Motivation and approach

As information systems become complex, the need for a highly-structured and flexible methodology becomes mandatory, since traditional approaches [10] result to be ineffective when applied to non-conventional cases such as the modeling of advanced inter-organizational scenarios. Inspired from these considerations, in this chapter we propose an innovative process-driven methodology for continuous information system modeling, which encloses a number of aspects of the information system life-cycle, from planning to implementation, and from usage to re-engineering.

Our methodology [103] is basically based on software planning and development, and it can be considered as a reasonable alternative to traditional proposals based on the Waterfall Model [18]. Similarly to lightweight and agile software development patterns [1], this methodology adopts iterative procedures, and it is characterized by short recurrent steps that are target-oriented and suitable to support an adaptive evolution of the whole information system modeling phase.

In our methodology, software planning and development is modeled via specifying two phases, directly connected to the concepts of process and function. This resembles the well-known Feature-Driven Development approach [11], which proposes a planning and implementation incremental procedure oriented to handle

homogeneous groups of functions. Our methodology integrates this approach with an analysis of processes that constitute the scenario in which functions execute.

Software planning and development tasks are composed by two macro-phases. In the first phase, we produce a scenario analysis adopting a Process-to-Function (P2F) approach, where we capture interactions among organization, information and processes. Specifically, in this phase we model the decomposition of processes in sub-processes and activities, and connect them to required information contents and specific portions of the business organization related to them. In the second phase, we produce a requirement analysis adopting a Function-for-Process (F4P) approach, where information system development is modeled, planned, and dynamically reported according to a package-oriented organization. Specifically, features of the information system are grouped in packages and integrated with processes that characterize the target scenario.

After the implementation and enactment of the information system, logs of executions are stored and analyzed by process mining techniques (e.g., [17]), which aim at extracting useful knowledge from traces generated by processes of at-work information systems. This way we are able to deduce paths of process executions, and, as a consequence, we can produce an ex-post analysis of scenarios, thus highlighting differences due to diverse execution scenarios of the realized information system. This analysis should be used as input of a new instance of the methodology for modeling a new (similar) information system, or re-engineering the actual information system.

The methodology we propose is supported by UML diagrams [9] that are able to offer a meaningful expressivity to developers and consultants, and an immediate interpretation to customers. In more detail, in the P2F phase, we adopt an approach based on Use Case and Activity Diagrams in order to obtain a Business Model characterized by a process-oriented representation. In the F2P phase, we still adopt Use Case Diagrams to model the logical structure of functions, but progressively we introduce Deployment Diagrams and Class Diagrams to model the architecture of the information system and its physical elements, such as databases, control modules, and interfaces. As a result of this phase, we obtain a Conceptual Model allowing us to define the functional analysis, and an Implementation Model allowing us to formalize requirements for the final realization of the information system.

3.2 Related works

The strict relationship among business processes and information systems has been firstly recognized in [31] at early 90's. Business processes heavily influence the final structure and functionalities of information systems. Symmetrically, the development of the information system influences the design of specific business processes of the target organization.

According to this evidence, several information systems modeling methodologies that, like ours, are focused on processes have appeared in literature recently. Also, some interesting applications of this novel class of methodologies have been proposed. Among such applications, we recall: (*i*) integration of process-oriented techniques and Data Warehouses [32], (*ii*) simulation of business processes to

precisely capture information systems requirements [33], (*iii*) process-driven modeling in the context of *e*-learning systems [34].

From the straightforward convergence of the mentioned research effort and practical applications, it is reasonable to claim that achieving a *total synergy* between design of business processes and development of information systems should be the goal of *any* organization, as stated in [35],[36],[37]. Nevertheless, in real-life organizations business analysts and information systems engineers very often have distinct roles within the organization, and, in addition to this, very often they use different tools, techniques and terminologies [38]. This contributes to make the achievement of the above-introduced synergy more difficult, and puts severe drawbacks with respect to a complete integration between organizations and information systems. On the other hand, it is very difficult to predict the "relative" consequences of changes occurring in business organizations and information systems [39], so that re-engineering issues play a critical role in this respect.

Giaglis [40] proposes an accurate taxonomy of business processes and information systems modeling techniques, also putting in evidence similarities and differences among the available alternatives. In [40], according to [41], the following perspectives of an information systems modeling technique are systematized: (*i*) *functional perspectives*, (*ii*) *behavioral perspectives*, (*iii*) *organizational perspectives*, and (*iv*) *informational perspectives*. As demonstrated throughout the paper, our proposed methodology strictly follows this paradigm, and meaningfully includes all the introduced perspectives, plus innovative amenities.

Implementation-wise, the methodology we propose is based on three levels of modeling and analysis, enriched with a final ex-post analysis of business process traces. Each level founds on classical UML diagrams enriched with *stereotypes* aiming at carefully modeling even-complex business processes by means of the so-called *UML Profiles*. This idea is not new in the context of information systems modeling techniques. For instance, in [42] a UML-based framework for modeling strategies, business processes and information systems of a given organization is proposed. Similarly to ours, this framework adopts a multi-level approach during the modeling phase. Other proposals based on the usage of specialized UML profiles for capturing several aspects of modeling information systems are [43,44,45].

Ex-post analysis of business process traces can be instead regarded as an innovative aspect of the methodology we propose. This resembles the work of Mendes *et al.* [46], where scenario evolution is modeled in terms of a specific process that captures organizational changes. Contrarily to this, in our methodology scenario evolution is not captured on the basis of a fixed, a-priori pattern, but instead it is deduced from the analysis of process traces originated by the interaction between users and the system.

Another distinctive feature of our methodology is the idea of separately modeling the *static knowledge* (i.e., the knowledge modeled by means of Use Case and Class Diagrams) and the *dynamic knowledge* (i.e., the knowledge modeled by means of Activity Diagrams). This amenity if finally combined with the ex-post analysis illustrated above, thus allowing us to achieve a powerful tool for mining and reasoning on processes, and, consequentially, significantly improving the modeling capabilities of the proposed methodology.

3.3 Scenario analysis and the Business Model

The definition of business processes characterizing the scenario in which the information system will operate is a milestone of the planning phase. Business Model is thus an essential input to the subsequent selection and definition of functions able to manage information useful for the specific information system context.

Scenario analysis is obtained as a result of the study of the target organization, interviews to members of the organization, reading of documents, selection of relevant procedures etc. All these elements are referred and represented in the Business Model, which is a formalization of organization processes, *actors* of the organization, and information. To efficiently support this formalization, Business Model is organized in several components: (*i*) *Process Schema*, which models processes of the information system; (*ii*) *Actor Schema*, which models actors of the information system; (*iii*) *Archive Schema*, which models *archives* of the information system.

Actors and archives are formalizations of *active* and *passive* entities that interact with processes. We represent them by adopting *stereotypes* of the native UML actor element. We consider as actors all the operators (human or automatic) that activate or enact a process. An archive is instead every information source useful for the execution of a process. In the actor and archive schemas, we model and represent taxonomies and ontologies [16] of entities in order to permit a meaningful *contextualization* of organization and information elements.

In the Process Schema, processes are modeled by means of a *top-down approach*. Specifically, we analyze processes and then select sub-processes that characterize each of them. Distinguishing between processes and sub-processes is a non-trivial engagement, which also strongly depends on the particular application context. In our methodology, in order to cope with this conceptual aspect we assert what follows.

A process is a set of procedures that are finalized to obtain a goal, starting from the input. A process involves a number of actors, and requires information modeled in terms of archives. Finally, a process is composed by sub-process.

A sub-process is an element of a process *P*, more restricted than *P*, but having the same formalization. A sub-process models components required for the release of a sub-service (or sub-product) of the information system. These components are referred as the path of execution of the sub-process. Finally, a sub-process can be *structured*, i.e. composed itself by other sub-processes in a hierarchical fashion, or *atomic*, i.e. without any sub-sub-process (in this case, the sub-process is named as *activity*).

An activity is an atomic element that represents a specific *portion of work*, and constitutes a logic step within a process. To model evolution of activities within a same process *P*, we make use of an *Activity Diagram* (see Fig. 9) that establishes the temporal order of the activities during the enactment of *P*.

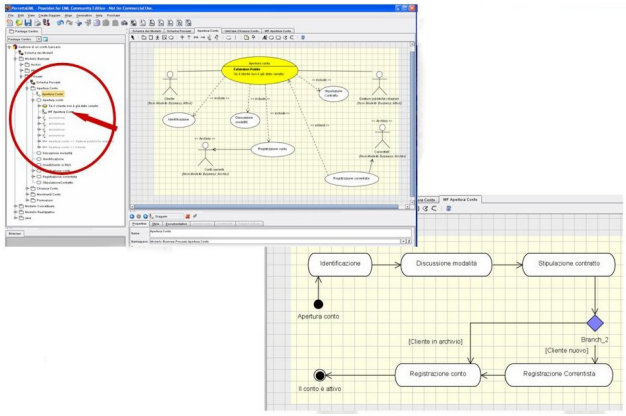

Fig. 9 From a Use Case Diagram to the related Activity Diagram

Top-down analysis focuses on high-level processes characterizing the information system scenario. In the visual representation implementing such analysis, we introduce a package for every macro-process. Each package contains a Use Case Diagram in which the use-case element corresponding to the process P is connected with use-case elements corresponding to every sub-process P_i of P. To model these connections, we use *include*, *extend*, and *specialize* associations provided by UML. In more detail, we assume the following association semantics.

A process P "includes" a sub-process P_i if, in every instance of P, an instance of P_i is required to be executed. A sub-process P_i "extends" a process P if, in every instance of P, an instance of P_i is executed only if a given condition is verified (this condition is expressed by the so-called *extension point* element). A sub-process P_i "specializes" a process P if P_i involves all the sub-processes involved by P and other specific activities.

A UML *association* is used to connect a use-case representing a process P or a sub-process P_i to an actor A or an archive S. Therefore, we are able to express that an actor executes a process (or a sub-process), and that a process requires or modifies information contained in an archive during its execution.

For each process P, we then model the path of execution of its sub-processes, via associating an Activity Diagram to P. As a consequence, we finally obtain that in the Use Case Diagram of P we represent a first analysis about the composition of P, and in the Activity Diagram we formalize the sequence of execution of activities of P and express pre-conditions and post-conditions among activities via conventional join, fork, and merge constructs.

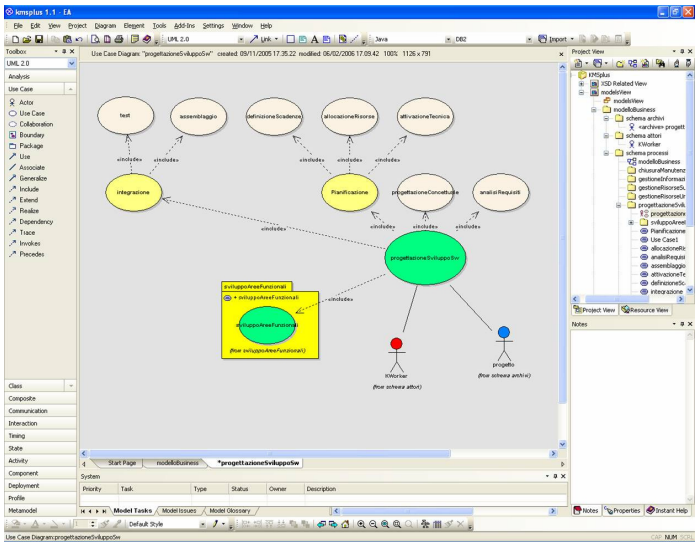

Fig. 10 Modular representation of processes

This decomposition is replicated for every sub-process that is itself a structured (sub-)process. To this end, we select its sub-sub-processes and connect them to it by means of *include*, *extend* or *specialize* associations. Then, we model the dynamic of the evolution of the sub-process by linking it to a specific Activity Diagram.

In total, for each process P, we introduce an Activity Diagram containing sub-processes P_i directly connected to P; furthermore, if a sub-process P_i itself involves sub-sub-processes $P_{i,j}$, their sequences of execution should be represented by another Activity Diagram connected to P_i. If a sub-process P_i of a process P is too much articulated to be represented in the main Use Case Diagram (of P), we introduce a sub-package B_i that contains another Use Case Diagram. This allows us to obtain a modular and incremental process organization that gives us benefits at both modeling and visualization tasks. In the main Use Case Diagram, we represent the sub-package B_i and its related sub-process P_i, and we connect B_i to P. The result is a hierarchical and modular representation of processes (see Fig. 10), that can be easily modified in a specific portion without conditioning the whole structure of the model. It should be noted that this amenity plays a critical role in complex and scalable information systems.

3.4 Analysis of function and the Conceptual Model

Scenario analysis describes the context in which the information system will operate. The next step is to analyze and model functions supported by the system in order to facilitate the execution of processes within the organization. Conceptual Model is the output of this phase. In the Conceptual Model, we provide: (*i*) a formal schema of functions and users; (*ii*) a formal schema of data; (*iii*) a formal schema of interactions between functions and data. Furthermore, Conceptual Model also represents functional blocks and views on data (i.e., information sources' schemas).

Functional blocks are modeled by use-case packages and taxonomies of actors, according to an approach similar to the one used to model processes in the Business Model (see 3.3). Data views are instead represented by means of Class Diagrams. Therefore, we can state that Conceptual Model is characterized by two aspects that capture the overall knowledge of the information system: (*i*) *static analysis* given by the *Data Schema*, which describes schemas of information sources, and *View Schema*, which describes views on the latter schemas; (*ii*) *dynamic analysis* given by the *User Schema*, which models users, and *Function Schema*, which models functions. Both static and dynamic analysis concur to capture even complex aspects of the information system, thus adding novel and useful amenities to traditional design methodologies.

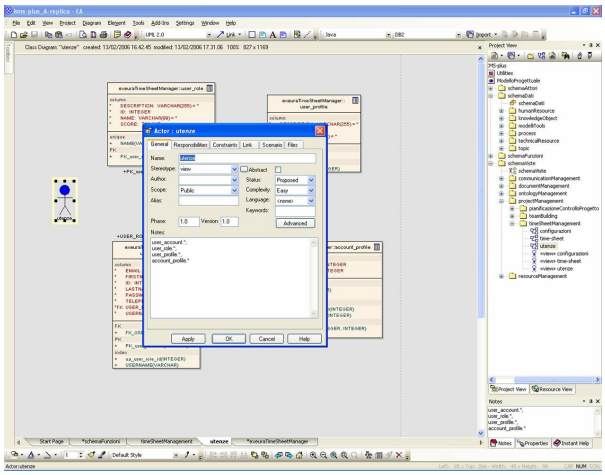

Fig. 11 A view and its documentation

Data Schema contains a Class Diagram that represents a conceptual model of the database underlying the information system. We use *Table* and *Key* stereotypes to adapt UML classes and attributes, thus modeling a data schema. Foreign keys and cardinality constraints are instead represented via UML associations among classes. At this level, we make use of *composition* and *aggregation* associations, and taxonomies to represent logical relations among entities. Therefore, Data Schema is a *high-level description* of the information system database, which is then detailed in the Implementation Model.

View Schema is a package that contains a Class Diagram named as *View Catalogue*. A view is a portion of database useful in a specific functional context. Each view is represented by a package containing a Class Diagram in which the involved-by-the-view entities/classes of the database are shown, along with their relations. In each package, a view is represented as an actor with the stereotype *View*, and can be exported in the Function Schema to model in detail the interaction among functions and data they require or modify. Also, we associate a documentation to each view *V* (see Fig. 11), such that this documentation contains additional information on *V* like: the logical name of *V*; for each entity/class, the list of specific attributes – obtained as a selection of the whole set of attributes – that are useful in the specific

functional context; the way used in the specific context to navigate associations among entities/classes etc.

User Schema has the same syntax of the one relative to the Actor Schema in the Business Model (section 3.3). While actors are entities (human or automatic) that activate or enact processes, users are instead entities (human or automatic) that will interact with the information system in the real-life realization.

Similarly to users, functions in the Function Schema are modeled by adopting syntax analogous to the one employed in the Business Model to represent processes, with the difference that archives are substituted by views (coming from the View Schema).

3.5 The implementation model and the development of the information system

Once requirement analysis is completed and Conceptual Model is defined, a physical planning of the information system is necessary. Guidelines defined in the Conceptual Model are mapped on the software architecture designed for the system. On the basis of the specific information system, different architectural solutions can be chosen, but every choice should include *at least* three levels of implementation: (*i*) a *Database Level* to model information/data sources of the system; (*ii*) a *Control Level* to models (software) classes implementing the application logic of system procedures; (*iii*) an *Interface Level* to model forms handling the interaction between system and users (human or automatic).

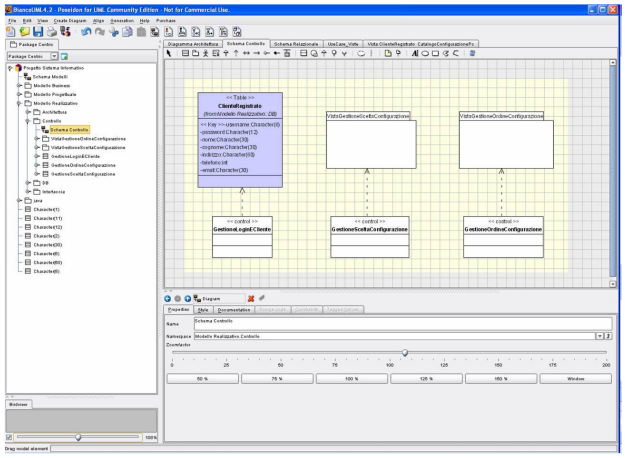

Fig. 12 A control schema

In order to efficiently support these requirements, Implementation Model is constituted by several components: (*i*) *Architecture*, which contains a representation of physical elements of the information system (i.e., the software architecture of the system); (*ii*) *Database*, which implements the Database Level; (*iii*) *Control*, which implements the Control Level; (*iv*) *Interface*, which implements the Interface Level.

Similarly to other models of our methodology, each component is implemented by a package, according to the following organization. Architecture component contains a Deployment Diagram where nodes and components of the system implementation can be defined. Furthermore, just like other constructs of our methodology, it is possible to define sub-packages in order to obtain a modular representation of the system implementation. Database component contains a Class Diagram enriched by stereotypes, named as DB Schema, which allows us to represent the schema of data stored in the information sources of the system (i.e., the database underlying the system). With respect to the Data Schema of the Conceptual Model, in the DB Schema of the Implementation Model we model in detail all components of data tables (e.g., attributes with data type and range of validity etc), in a similar way to what happens in conventional CAD tools for E/R diagrams, thus obtaining a linear description of the system database. Control component contains a Class Diagram, named as Control Schema (see Fig. 12), in which a control class's catalogue is represented. Each control class is implemented as a UML class with stereotype Control, and contains methods used by the Interface Level to manage data from the Database Level. Also, each control class refers to one or more views inherited from the DB Schema on the basis of their relevance and scope with respect to the specific functional context. Methods of each control class are described within the UML class in forms of software interfaces (e.g., Java-based) and documentation in free text.

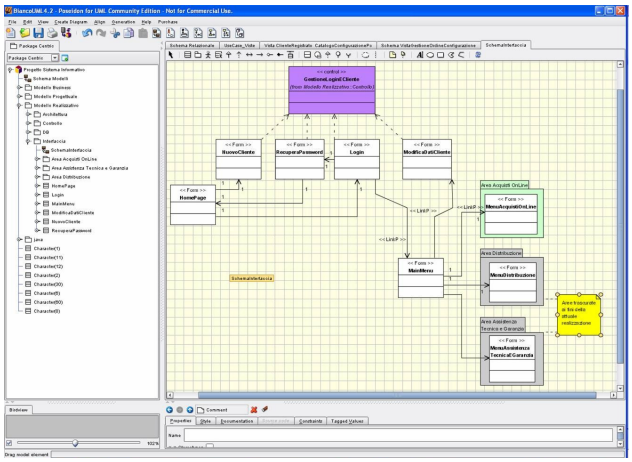

Fig. 13 An interface schema

Following the organization of the Implementation Model, Control Level is invoked by the Interface Level that contains a Class Diagram, named as *Interface Schema* (see Fig. 13), which models the interaction between system and users. Paths of interactions are expressed in Activity Diagrams of the Conceptual Model; we use these paths to model a sequence of forms, which, in our implementation, are UML classes enriched with specific stereotypes. Specifically, a form is characterized by three elements that determine the final representation of such form (see Fig. 14): (*i*) *entry unit*, which is an area of the form where users submit input elements to the system via traditional GUI controls such as text fields, combo boxes, check boxes etc; (*ii*) *data unit*, which is an area of the form where information derived from the

underlying database (i.e., sets of tuples) is shown; (*iii*) *display unit*, which is an area of the form where static components are shown (e.g., textual information describing how to use form controls).

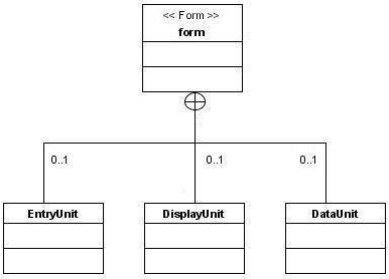

Fig. 14 Conceptual representation of a form

In our methodology, a form can be a *plain form*, a *list form*, or a *recursive form*. Plain forms are basic realizations of the form construct. List forms, modeled by the stereotype *FormL*, are used to represent forms in which sets of tuples are shown. Recursive forms, modeled by the stereotype *Form**, are used to represent forms that are shown many times, one for each tuple corresponding to a specific parameter.

When forms transmitting parameters to other forms are considered (e.g., during user transactions), we support this facet of the information system by appending specific attributes to UML classes. These attributes are described by the stereotype *LinkP*. To ensure data consistency, we simply impose that the type of transmitted parameters is the same of (appended) attributes in the related UML class.

3.6 The application scenario

The methodology we propose is actually experimented in *Exeura* [15], a spin-off company of the University of Calabria that operates in the IT and KM (*Knowledge Management*) areas.

Given an application domain, our methodology should be executed in a *complete* or *partial* way, by specifying and personalizing the P2F and F4P phases on the basis of the specific context. P2F phase is executed according to an agile approach, characterized by short and adaptive iterations, each of them containing an analysis step and a modeling step. Frequent interactions with customers allow validation and revised versions of the analysis, and, if needed, inclusion of novel elements useful to support new procedures of the information system. F4P phase is instead executed according to an incremental approach. A first architectural and functional model of the system is delivered, so that every system procedure is supported by a specific functional analysis and an implementation phase, where a high-level of concurrency and feedback is permitted. In particular, Implementation Model is structured in such a way as to enable reverse engineering methods [14] starting from the source code.

If the whole software planning and implementation must be executed to realize a new system, P2F and F2P phases are both required, of course. In this specific case, thanks to process mining techniques that allow us to analyze traces of execution of

processes (e.g., [17]), is then possible to deduce an ex-post evaluation of the scenario that in turn generates a new instance of the methodology. Here, the goal is to take advantages from previously-modeled scenarios when novel, similar scenarios are considered. If the re-engineering of an existing system is required, scenario analysis should be derived through the analysis of process logs generated by process mining tools such as *ProM* [13].

Part II

The advances

4

Ontology for modelling business process knowledge

Summary: This chapter describes an ontology-based organizational knowledge representation framework focused on the specification of a two kinds of ontologies: the top level ontology containing concepts characterizing the typical organizational background and COKE ontologies representing so called *core organizational knowledge entities*. The framework constitutes an abstract representation of organizational knowledge providing a semantic support for designing knowledge management infrastructure able to interoperate with systems already existing in an organization. Moreover, the annotation of COKE w.r.t. the top level ontology allowed by the framework facilitates their semi-automatic handling, retrieval and evolution monitoring.

4.1 Information systems and organizational knowledge

In the last years many enterprise models aimed to give a formal representation of the structure, activities, processes, information, resources, people, behaviours, goals, and constraints of a business, government, or other enterprise has been proposed in literature [4]. All these models consist of an ontology based on a vocabulary along with some specification of the meaning or semantics of the terminology within the vocabulary. For example, the Toronto Virtual Enterprise Ontology (TOVE) [3] is an ontology providing a shared terminology for the enterprise that defines the meaning (semantics) of each term in a precise and an unambiguous as possible manner using first-order logic; IDEF Ontologies [2] intended to provide a rigorous foundation for the reuse and integration of enterprise models; CIMOSA [5] aimed to provide an appropriate integration of enterprise operations by means of efficient information exchange within the enterprise with the help of information technology.

All these ontologies attempt to describe in detail the whole organizational knowledge and structure. The resulting models are less flexible and not easily applicable in the very dynamic contest of a real enterprise.

This chapter describes an ontology-based framework for specifying organizational knowledge. The framework aims to represent so called Core Organizational Knowledge Entities (COKE) in an ontology expressed using a novel ontology representation language based on disjunctive logical programming. The Framework is organized as a two level family of ontologies: the first level (top level) ontology represents the set of concepts characterizing organizational background; the

second level ontologies formally represent the COKE (i.e. human resources, business processes, technical resources, knowledge objects). The resulting formal representation of organizational knowledge aims at contributing as theoretical base in supporting the analysis and design of Knowledge Management Systems (KMS) in two manner: first ontologies represent an abstract representation of organizational knowledge providing a semantic layer allowing interoperability between existing systems and the KMS, second COKE can be easily annotated w.r.t. top level ontology using semi-automatic mechanisms, so their evolution can be better captured and handled.

4.2 The Ontology-Based Framework

The proposed ontology-based framework [6] is organized as a set of ontologies as described in Fig. 15. The top level ontology (topic ontology) contains concepts characterizing organizational background knowledge. These concepts are used for annotating COKE.

The COKE ontologies formally represent human resources, business processes, knowledge objects, technical resources constituting the main elements characterizing the organizational structure and playing a fundamental role in business activities execution. All the ontologies are strictly connected by relations between their own elements and are represented using the DLP+ language.

The framework give an abstract representation of COKE's allowing semantic interoperability among the various type of information systems used in the organization. More in detail, the framework provides a uniform representation of knowledge handled by the systems already existing in the organization such as document and project management systems, ERP and CRM systems. Connecting such systems to the framework using ad hoc software modules handled knowledge object can be better stored, managed and retrieved.

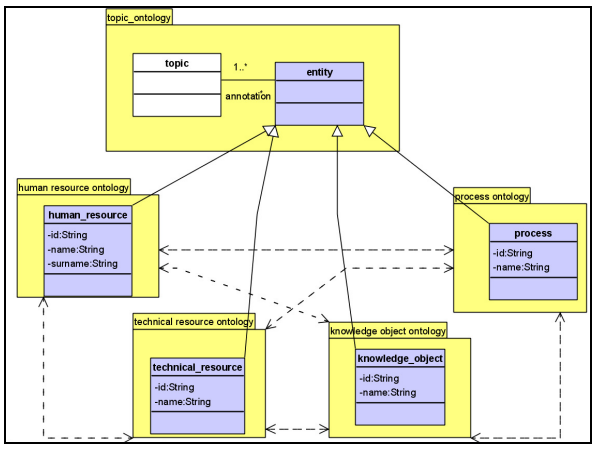

Fig. 15 The organizational knowledge framework

4.2.1 The Top Level Ontology

The top level ontology or topic ontology contains concepts characterizing the typical organizational background. It specifies the explicit and implicit organizational declarative knowledge concerning the concepts characterizing an application domain: e.g. an IT enterprise background is founded on concepts coming from computer science field. As top level ontology it provides the other COKE ontologies with concepts to formally annotate their contents.

4.2.2 The COKE Ontologies

COKE Ontologies contain the formal representation of human resources and their organization in groups, processes and their activities, knowledge objects constituting elements produced or used in business processes, technical resources in term of instruments used during business process execution.

The Human Resource Ontology represents individuals working in the organization (knowledge workers) and social groups they are involved in. Each individual profile is represented in term of implicit, explicit, individual and social knowledge, organizational role, social group membership, required technical resources. Each social group (community of practice, project team, organizational group, etc.) profile is represented in term of its members profiles.

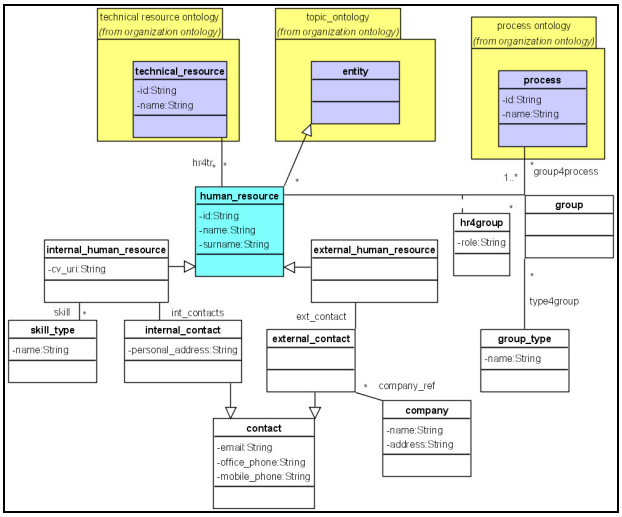

Fig. 16 The Human Resource ontology

The Business Processes Ontology contains procedural knowledge related to the managerial, operational and decisional processes. Each of them is described in terms of activities, sub-processes, transition states and conditions, involved actors, treated topics, etc. This can be a simple representation of business process or a complex ontology where the workflow structure and the taxonomic and non-taxonomic relations between processes are represented using the DLP+ language [Leone, 04].

The business process ontology exploits an interesting capability of DLP+ language allowing the expression of relations between classes, that enables the representation of process meta-model, process schemas and process instances.

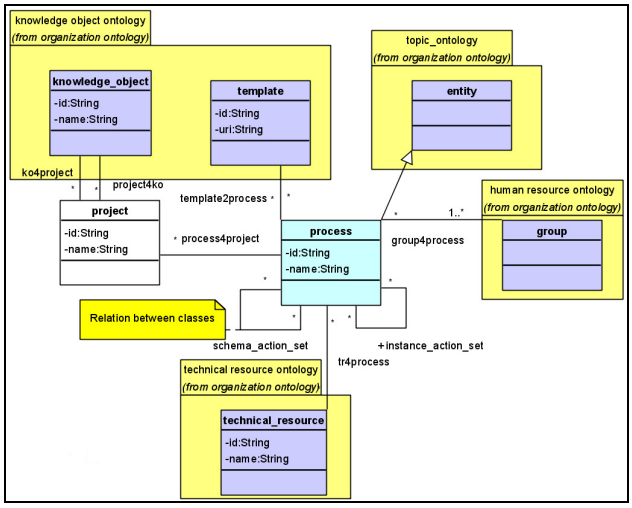

Fig. 17 The business process ontology

The Knowledge Objects Ontology maps the structure of logical objects (e. g. database schema, database tables, textual documents, web pages, etc.) containing explicit knowledge under structured, semi-structured or unstructured form [1]. These are used in the business processes and handled by the human resources through knowledge-based tools. Knowledge objects retrieval, management and handling is facilitated by the annotation on the topic ontology concepts.

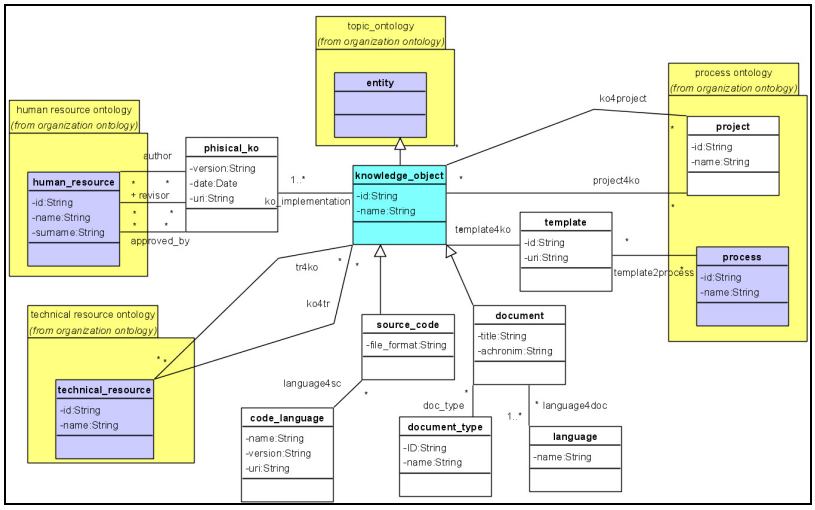

Fig. 18 The business object ontology

The Technical Resources Ontology identifies the tools by which knowledge objects are created, acquired, stored and retrieved. The execution of a query to the top level ontology can be executed using a specific tool able to retrieve all the elements related with a specific concept. Element can be filtered to obtain a specific COKE related to the query. For example a query result can contain people knowing a given concept or systems containing knowledge objects related to some concepts. This allows the management of implicit and explicit knowledge stored in structured, semi-structured or unstructured machine-readable forms.

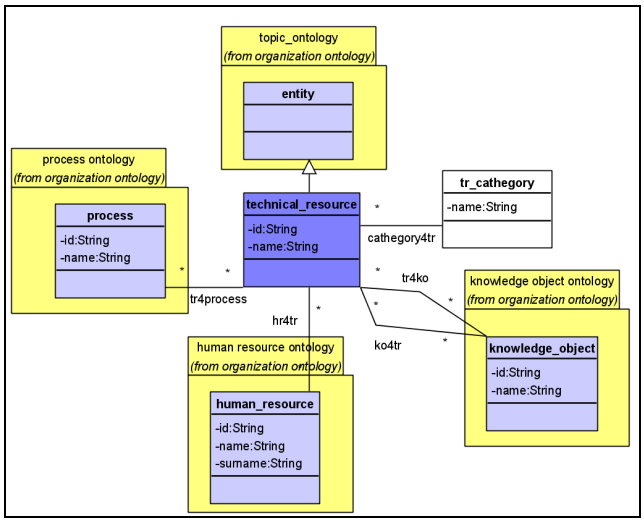

Fig. 19 The technical resource ontology

4.3 Future enhancements

This chapter describes an ontology based framework for organizational knowledge representation providing an abstract definition of COKE's enabling the dynamic capture of business processes changes and evolutions. Moreover the framework allows the automatic annotation of COKE's to enterprise relevant concepts allowing semantic retrieval and management capabilities. Future works regard the definition and implementation of the annotation mechanisms and the representation of time in the ontology.

5

Ontology for process oriented Information Systems

Summary: An ontology-based approach for supporting Loosely-Structured Cooperative Processes (LSCP) is defined in this chapter. Ontology are adopted to provide an enterprise contextualization of operations executed on heterogeneous context by indipendent component integrated in Service Oriented Architecture. A semantic analysis of the domains of these operations offers an input to process mining algorithms that are able to discover procedural knowledge about enterprise.

5.1 Loosely-Structured Cooperative Processes

The "internetworked" enterprise domain is a challenging application scenario, due to the complexity and dynamicity of the collaboration processes that typically arises in such a context, which could greatly benefit from some suitable elicitation, management and sharing of both intra- and inter-organizational information and knowledge. In particular, it is desirable to provide both workers and decision makers with a unified and high-level view over the cooperation processes. This usually requires quite long and complex analysis tasks, but can be supported by knowledge discovery techniques, devoted to extract and restructure knowledge about the processes, as well as to guide and optimize future cooperation work. Many possible scenario are considered in literature. For example, an approach for grid-oriented virtual enterprise is proposed in [101].

Let now consider a context in which heterogeneous functional component of a distribute informative system are running and each one operates on a specific semantic context. Basing on a synchronization protocol and on a mapping schema of local data, an architecture inspired to the SOA protocol is able to provide an integration of these components. We propose a knowledge-based framework for supporting an ex post analysis of the operations executed by the components finalized to extract knowledge about dynamic aspects of the enterprise.

Our goal is to offer a contextualization of the operations executed on different semantic context. Adopting an enterprise view, local operations and data are framed as steps and parameter of so defined "loosely-structured cooperative processes" (*LSCP*). In fact, emerging work models increasingly take the form of loosely structured, often self-organising networks of nimble and virtual knowledge work

teams within and between organisations. So, with an appropriate semantic interpretation of log of execution captured on a service oriented architecture, it is possible to retrieve contextually relevant knowledge elements in order to elaborate process schemas representing collaborative and distributed work.

To obtain this goal, our approach provide an enterprise model derived as an extension of the one illustrated in the previous chapter and applied on an architectural scenario characterized by an Enterprise Application Integration protocol based on message interchanging. We propose a methodology to elaborate the logs contained in the messages, basing on the operations' domain that is composed by either document, user or project unit. Focusing on each element we apply process mining algorithms to obtain process schemas representing dynamic evolution of the domain [104].

We apply this framework to an enterprise project-oriented context, considering an information system composed by functional component providing project management, content management, user management and timesheet management.

5.2 A framework for supporting and tracking LSCPs

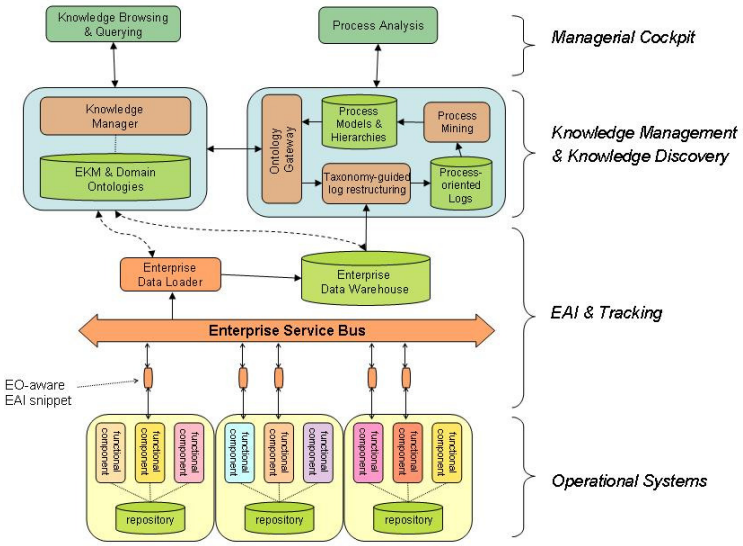

Fig. 20 Conceptual architecture of the framework

Our purpose for loosely-structured cooperative processes analysis is based on a multi level framework, as illustrated in Fig. 20. The functional capability to support the enterprise requirements are provided by the operational components that are involved in the distributed information system. An interoperability among components is provided by an Enterprise Architecture Integration level: every functional module is enriched by an *EAI snippet* able to capture events generated by

local operations and frame them in an enterprise context. EAI snippets generate messages that contain information about the operation and its domain. These messages are captured on the Enterprise Service Bus that represents the connection among component. An ontology-based Enterprise Data Loader (EDL) provide to extract knowledge about operations with repsect to a shared enterprise model and this knowledge is stored in a centralized Enterprise Data Warehouse (EDW).

The EDW constitutes the knowledge base for the Knowledge Managment and Discovery level, that is composed by two features that allow ontology management and process analysis features, available on the managerial cockpit provided by the highest architectural level. Enterprise ontologies are invoked by the EDL module to extract knowledge contained in the messages transmitted on the ESB. They are also useful to examine the operations' domain to individuate elements on which effectuate the log reconstructioning. The process logs are captured with respect to different level of process abstraction: taxonomies of processes allow to consider more or less detailed descriptions of operations' domain contained in the EDW. The output of this elaboration is in a Process-oriented Log Data Base and so processed by a Process Mining module. The result offered to enterprise manager is new process schemas, that can be stored in the enterprise knowledge base to refine process taxonomies and contribute to the discovering of new knowledge patterns.

5.3 The Enterprise Integration and Tracking level

5.3.1 The Enterprise Service Bus

During the 1990s, companies bought packaged software solutions for the enterprises that worked well individually but they created information islands. In most cases, each system produced redundant information (like customer information) and the process of manually updating the related informations in different systems quickly becomes cumbersome. Eventually, some of the data across systems became inconsistent.

From problems like double data entry, inconsistent data, and data isolation the "enterprise application integration" (EAI) was born, to combine separate applications into a co-operating federation of applications. The subsequent picture shows three different pattern of EAI for interoperability purposes adopted in the last decades. A time-line, from left to right, is implied.

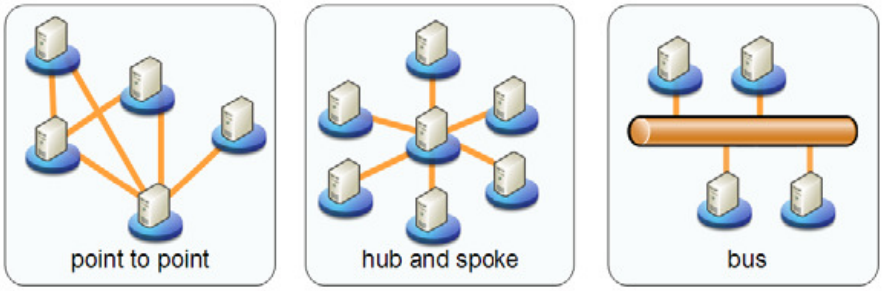

Fig. 21 Enterprise Application Integration patterns

Firstly, developers adopted point-to-point integration because it was easy to understand and quick to implement. As additional applications are integrated in a preexisting set, the situation becomes unwieldy. Each application is tightly coupled with the other ones through their point-to-point links. Changes in one application may break the correlated applications. Another disadvantage with this approach is the number of integration points needing support. If we have five applications integrated with one another, we need ten different integration points. As a result, each additional application becomes harder to integrate and maintain.

To avoid this problem an intermediate layer to isolate changes in one application from the others is necessary. This layers are often called “integration middlewares” and they was originally built mostly around proprietary systems with their own canonical protocols and data formats operating in centralized hub configurations. To integrate a system into the hub some adaptors to convert its protocols and data formats to the hub’s canonical protocols and data formats are necessary. The protocol and format conversions results also in low overall performance. Support for standards, if any, was often tacked on to EAI products as barely working afterthoughts.

A very popular middleware technologies are those called “message-oriented middleware” (MOM). Applications communicate with one another by passing messages and these messages are queued if the receiver is unavailable thus guaranteeing that the messages will eventually be delivered.

The Enterprise Service Bus (ESB) is a new kind of middleware that supports services-oriented interactions among enterprise applications. In the ESB model, most or all applications and services in the enterprise connect to the ESB and communicate with each other over the ESB. Applications and services usually connect using SOA standards, whereas legacy systems require integration via traditional EAI technologies such as adapters. The communication between endpoints is handled by message oriented middleware: programs connect to the ESB and send or receive messages. The ESB handles routing details, mediation of differences between endpoints, and the physical details of communication.

While the ESB can be described as a recent approach to enterprise connection and integration, it stands firmly on the shoulders of three disciplines that have already proven their worth in the enterprise. The ESB concept is made possible through the convergence of Service Oriented Architecture (SOA), Enterprise Application Integration (in traditional sense), and Message Oriented Middleware. The rapid

progress being made in each of these disciplines is all leading in the same direction, convergence.

There are multiple complete definitions proposed for an ESB. However most definitions encompass a healthy subset of the following composite definition.

- An ESB is a backbone for connecting and integrating an enterprise's applications and services.
- An ESB provides the necessary infrastructure to create a service oriented architecture.
- An ESB is a convergence of EAI, MOM, and SOA concepts.
- An ESB is based on open standards such as XML, SOAP, and WS-*.
- An ESB provides intelligent routing, such as publish-subscribe, message brokering, and failover routing.
- An ESB provides mediation, overcoming data, communication, and security differences between endpoints.
- An ESB integrates with legacy systems using standardsbased adapters.
- An ESB provides logical centralized management but is physically decentralized.
- An ESB is able to apply EAI concepts such as rules and orchestrations.
- An ESB is able to monitor and throttle activity as per a Service Level Agreement (SLA).

In our framework interoperability between software modules at the operational level was well satisfied from the capabilities of an open source ESB named "Mule". It has been enriched by specific services that allow registration/unregistration of modules, sending/receiving messages. Also "internal" services was implemented to provide fully configurable routing capabilities, message dispatching to previously registered modules, return receipt dispatching. The subsequent picture shows our experimentation scenario that involves four operational modules for project, time sheets, resources and contents management.

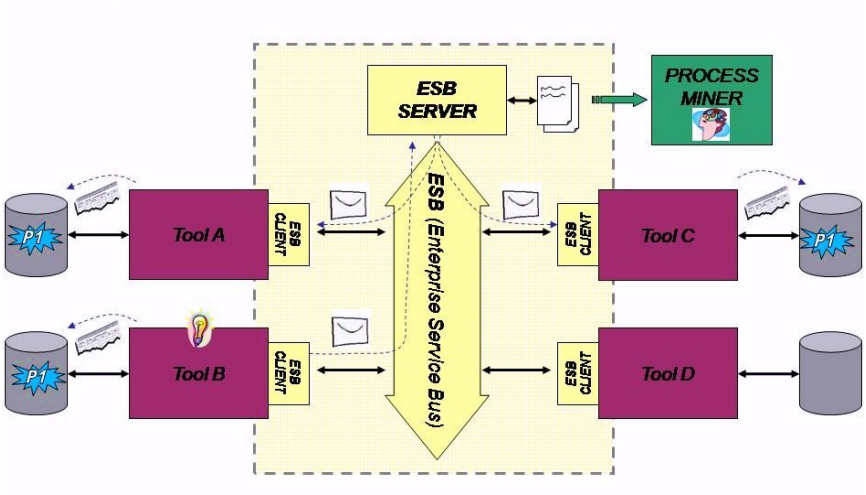

Fig. 22 ESB experimentation scenario

5.3.2 The Enterprise Knowledge Model

The data level shared by different functional context constitutes a semantic model defined as Enterprise Knowledge Model (EKM). It is an extension of the COKE level belonging to the framework introduced in section 4.2.2. It represents an abstract view on which the data structures characterizing the specific functional area integrated by the architecture are mapped. A detail of the semantic relationships between each data structure and the EKM is provided in a *mapping registry*. EKM entities are organized by domain. Since our purpose is to support process-oriented organizations, we consider two relevant domains:

- a "user" domain, containing a description of human resource, roles and workgroups
- a "process" domain, containing a description of tasks, assignments to the human resources, contents, technical resources and workspaces

We define *Enterprise Operation* (EOp) each operation that is enacted by one of the tools integrated by the architecture and that is able to produce an effect on one of the entities belonging to the EKM. So, for example, the creation of a new project in a project-management tool should be receipt by a content-management tool as a creation of a specific workplace. Therefore, an EOp is an event that is expected in a specific functional context but it is also relevant as a step of a collaborative process.

Traditional approaches to business process management impose to model a procedural schema before individuating tools or services that are able to perform each step. This implies that identifying which parts of business are supported by which parts of the system is not a straightforward task [62]. Our approach to process modeling, based on EOps, is instead characterized by an ex post analysis of the usage pattern of each functional component that produces an effect on the shared semantic model: by analyzing the effects of the local events on the EKM, we are able to frame heterogeneous operations as steps of a unique distributed process and so, by applying

process mining algorithms to the logs of execution of these operations, we are able to obtain a schema of the process.

EOps are so generated consequently to a trigger produced by a tool: an EOp should be activated by different tools and each of them give raise to the propagation of the effects of the operation on the other tools. The synchronization of the tools is provided by the architecture illustrated in the previous section. Knowledge maps are able to frame local characteristics of operations in a more abstract enterprise view: they correlate functional data structures to the EKM entity to which they are referred.

5.4 Analysing process logs

Process mining techniques, recently appeared in the literature, are a valid means for automatically extracting new knowledge on the behavior of a process, based on data gathered during its past enactments and stored in suitable logs (see, e.g., [21] for a survey on this topic). Clearly, such ex-post analysis of process executions, make such techniques quite different from traditional "business process monitoring'' solutions, which primarily focus on performances aspects, and typically offer simple statistics and mechanisms for detecting problematic cases.

Different kinds of process mining techniques have been defined in the literature, which, based on historical log data, can extract abstract process models, according to different perspectives of the processes, such as, e.g. flow of work, social relationships. For the sake of conciseness, in the remainder of this section, we only focus on the mining of workflow schemas.

Although there is a plethora of languages for modelling a process, most process mining approaches focus on *workflow* models, which describes both process activities and routing constraints that coordinate their execution. As an illustrative example, consider the toy HANDLEORDER process for managing customers' orders in a company, shown in Fig. 23. Here, edges represent precedence relationships, while additional constraints are expressed via labels associated with activity nodes. For example, task l is an AND-join activity, as it must be notified that both the client is reliable and the order can be supplied correctly. Conversely, b is a XOR-split activity, since it can activate just one of its adjacent activities.

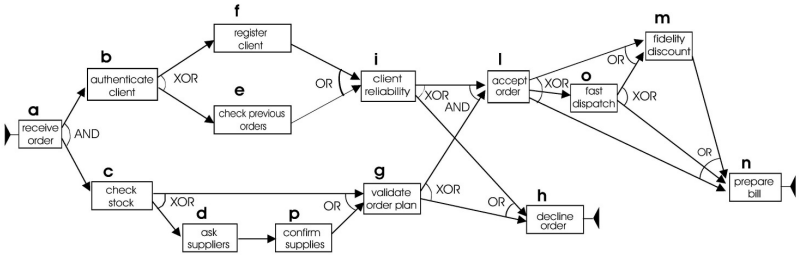

Fig. 23 Workflow schema for the sample HANDLEORDER process

Each time a workflow schema *W* is enacted, it produces an *instance*, i.e., a suitable sub-graph satisfying all constraints associated with *W*. Most process-oriented

systems typically store different kinds of events occurred during each process enactment. A popular format for representing log data is described next.

5.4.1 The MXML format for process logs.

MXML is an XML-based format recently introduced for representing event log data, which is now widely in the research community, and in the popular process mining framework *ProM* [98]. The structure of an MXML document is shown in Fig. 24. The root node of an MXML document represents a log file, with a *Source* element possibly indicating the system it was imported from. A workflow log can contain an arbitrary number of *Processes*, each of them collecting a series of executions for a specific process, which are modelled by different *ProcessInstance* elements. Each process instances consists of a number of *AuditTrailEntry*, each of which describes one specific log event. Every audit trail entry mandatorily contains two elements: the *WorkflowModelElement,* which indicates the workflow task the event refers to, and *EventType*, specifying the running state of the event, such as, e.g., scheduling, completion, suspension, termination.

Two optional elements of an audit trail entry are *Timestamp*, storing when the event occurred, and *Originator*, which identifies the resource, e.g., person, program or component, that triggered the event in the system.

Additional information can be stored in connection with different elements, by way of *Data* elements, consisting of attribute-value pairs.

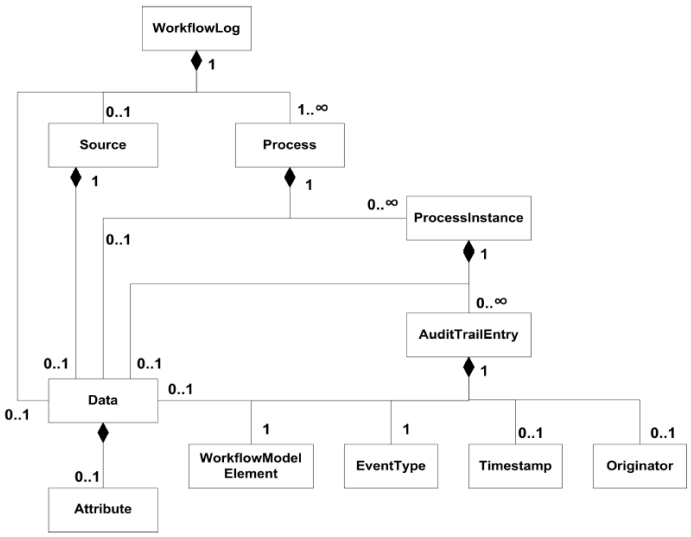

Fig. 24 The MXML format: a standard for process logs

5.4.2 Process Mining solution adopted

Many process mining approaches proposed in the literature [97] are aimed at discovering a single workflow model (like the one shown in Fig. 23), and mainly differ in the formalism used to represent workflow models.

The goal is to describe the events in a given process log (possibly represented according the MXML format discussed above) in an accurate and compact way. It is worth noting, that even such a model for the process already exists, the application of process mining can help to discover the real behaviour that occurs in the practice, possibly evidencing differences with the prescribed one.

Some recent proposals try to overcome the difficulty of these approaches in discovering precise models for processes exhibiting complex dynamics. In particular, the approach proposed in [97] extracts a set of workflows, collectively named *disjunctive workflow schema*, which provide a modular and accurate representation of the process. The approach implements a hierarchical, top-down, clustering procedure, where traces sharing a similar behaviour are clustered together, and then equipped with a specialized schema, possibly obtained by using some classical process mining algorithm. At the end of the procedure, a hierarchy of workflow schema is obtained, whose leaves constitute a disjunctive schema for the log. In order to efficiently partition a set of traces by well-known clustering methods, we resort to a "flat" relational representation of the traces, by projecting them onto suitable features, expressing behavioural patterns that are not modelled properly by the workflow schema that is being refined.

The approach sketched above has been recently extended in [96] by combining the clustering of process executions with ad-hoc abstraction technique, aimed at obtaining a compact and handy representation for each high-level schema, which emphasizes the most relevant behavioural features while abstracting from specific details. The result is a novel process mining approach capable of building a taxonomy of process models. In a nutshell, the taxonomy is modelled as a tree of workflow schemas, where the root encodes the most abstract view, which has no pretension of being an executable workflow, whereas any level of internal nodes encodes a refinement of this abstract model, in which some specific details are introduced. In other words, leaf nodes stand for concrete usage scenarios (computed through the clustering), whereas each non-leaf node (computed through abstraction mechanisms) is meant to provide a unified, generalized, representation for all the process models associated with its children.

Both these latter techniques, as well as previous algorithms, have been integrated in our current implementation of the architecture described in section 0515.2. In fact we believe that the discovery of hierarchies and taxonomies of process models can well support the creation of process ontologies, hence enabling for consolidating the knowledge on the different schemes of collaborative work adopted in the enterprise.

5.5 From LSCP log to process schemas

The application of process mining techniques to the execution logs gathered in a cooperation environment like the one sketched in section 05.2 is not trivial. Indeed,

process mining techniques found on the assumption that each registered event refers to a well-defined step in the process and to a case (i.e., a process instance). Conversely, such information might not be available in an unambiguous way in the case of loosely structured cooperation schemes, where different kinds of processes can be spontaneously arise without a well-specified model, and are carried out on the basis of elementary functions provided by the operational systems.

On the other hand, the availability of background knowledge concerning the organizational structure and the domains involved in the cooperation processes provides a valuable semantics-enhanced way to restructure such low-level logs into suitable process logs, where the log events are represented at some proper level of abstraction suitable for the analysis.

5.5.1 Abstraction-based restructuring of EOp logs

In the following we shortly describe the main kinds of information stored in the EOp logs collected in the cooperation architecture based on ESB.

A key concept in the collaboration scenario depicted so far is represented by the *Project*, intended as a bunch of (possibly not completely specified a-priori) activities, aimed at achieving specific targets and constraints, and associated with a range of resources. In a sense, projects are a sort of logical containers for the basic operations, which can be very useful for monitoring and analysis purposes.

Each atomic event registered in this log regards the execution of some Enterprise Operation. Different kinds of EOp exist that can be classified according to different perspectives, pertaining, e.g., their function and context. For instance, possible categories could be *project definition*, *user management*, *resource management*, *project run, content creation*.

EOP instances can be associated as well with information on the actual parameters of the operation executed. Clearly the class of major parameters strongly depends on the kinds of operation performed: e.g., *Users* in the case of *user management* operation, *Documents* for *content management* operations, *Tasks* for *task run* operations. Each operation instance refers to the *Tool* it was carried out with, and the *Agent* that performed it, and contains a timestamp.

Fig. 25 sketches an EOp log, registering a series operations performed in a toy example, concerning only two different projects, named *KMS-plus* and *PROMIS*.

#	tool	operation	subject	agent	project	timestamp
1	UM	createUser	Serge	Serge	default	2006.12.3009:30:00
2	UM	createUser	Dominic	Serge	default	2006.12.3009:30:00
3	UM	createUser	Lewis	Serge	default	2006.12.3010:35:00
4	UM	createUser	Edward	Serge	default	2006.12.3010:35:00
5	UM	createUser	Thomas	Serge	default	2006.12.3010:35:00
6	UM	createUser	Frank	Serge	default	2006.12.3010:40:00
7	UM	createUser	Mark	Serge	default	2006.12.3010:45:00
8	UM	createUser	Frederick	Serge	default	2006.12.3010:50:00
9	PM	creationProject		Serge	KMS-plus	2007.01.0108:30:00
10	PM	creationProject		Serge	PROMIS	2007.01.0108:30:00
11	PM	assignLeader	Dominic	Serge	KMS-plus	2007.01.0108:35:00
12	PM	assignLeader	Dominic	Serge	PROMIS	2007.01.0109:00:00
13	PM	assignResource	Mark	Dominic	KMS-plus	2007.01.0110:20:00
14	PM	assignResource	Mark	Dominic	PROMIS	2007.01.0110:20:00
15	PM	assignResource	Thomas	Dominic	KMS-plus	2007.01.0110:40:00
16	PM	assignResource	Lewis	Dominic	KMS-plus	2007.01.0110:50:00
17	PM	assignResource	Thomas	Dominic	PROMIS	2007.01.0110:50:00
18	PM	assignResource	Frank	Dominic	KMS-plus	2007.01.0111:00:00
19	PM	assignResource	Edward	Dominic	PROMIS	2007.01.0112:20:00
20	PM	assignResource	Frederick	Dominic	KMS-plus	2007.01.0112:21:00
21	PM	assignResource	Frederick	Dominic	PROMIS	2007.01.0112:22:00
22	PM	startTask	T1	Mark	KMS-plus	2007.01.0114:00:00
23	PM	startTask	T2	Mark	PROMIS	2007.01.0309:00:00
24	PM	endTask	T1	Mark	KMS-plus	2007.01.2014:00:00
25	PM	endTask	T2	Mark	PROMIS	2007.01.3114:00:00
26	PM	startTask	T3	Thomas	KMS-plus	2007.02.0114:00:00
27	PM	startTask	T4	Lewis	KMS-plus	2007.02.0214:00:00
28	PM	startTask	T5	Thomas	PROMIS	2007.02.0314:00:00
29	PM	endTask	T5	Thomas	PROMIS	2007.03.0214:00:00
30	PM	startTask	T6	Edward	PROMIS	2007.03.0814:00:00
31	PM	endTask	T3	Thomas	KMS-plus	2007.03.3114:00:00
32	PM	endTask	T4	Lewis	KMS-plus	2007.04.0214:00:00
33	PM	startTask	T7	Lewis	KMS-plus	2007.04.0409:00:00
34	PM	startTask	T8	Frank	KMS-plus	2007.04.0614:00:00
35	PM	endTask	T6	Edward	PROMIS	2007.04.1014:00:00
36	PM	startTask	T9	Frederick	PROMIS	2007.04.1514:00:00
37	PM	endTask	T9	Frederick	PROMIS	2007.04.2014:00:00
38	PM	startTask	T10	Dominic	PROMIS	2007.05.0409:00:00
39	PM	endTask	T10	Dominic	PROMIS	2007.05.0809:00:00
40	PM	endTask	T7	Lewis	KMS-plus	2007.05.2409:00:00
41	PM	endTask	T8	Frank	KMS-plus	2007.05.3014:00:00
42	PM	startTask	T11	Frederick	KMS-plus	2007.06.0309:00:00
43	PM	endTask	T11	Frederick	KMS-plus	2007.06.1214:00:00
44	PM	startTask	T12	Dominic	KMS-plus	2007.07.0409:00:00
45	PM	endTask	T12	Dominic	KMS-plus	2007.07.0809:00:00

Fig. 25 Simplified representation of an EO log for a test example

Notice that, for the sake of clarity, just one parameter, denoted as subject, is admitted to be involved in each operation instance. And just two tools, PM ("project manager") e UM ("user manager"), are considered able to perform EOp. In a real execution log, subject is a set of parameters constitutes of EKM entities involved in the EOp executed by tools. Every instance of EOp is characterized by a *process* attribute that constitutes a contextualization of the operation as step of a process. This is delivered by local tools, that in correspondence to an internal operation export towards the ESB the involved data and the context to which they are referred. In this example, *process* attribute is mapped on *project* entity, because all the data managed by the tools integrated in the ESB are framed with respect to a project view. In other words, if we consider an enterprise domain, *workspace* entity provided by a content management tool are correlated to *cost-area* entity provided by a timesheet management tool and both entities are connected to the *project* entity. In this context, every EOp performed by tools are so connected to data that refers an instance of the project entity. Mapping the project entity to the process attribute required by logs structure, we are able to offer a contextualization of the EOp.

Due to the scarce semantic content associated with the basic domain of EOp, this dimension of the log is not a good candidate for being mapped into the tasks (i.e.,

WorkflowModelElement) of the unknown workflow model that a process mining algorithm should eventually discover.

There can be different ways to restructured such logs into a process logs suitable, for the application of process mining techniques. The availability of ontologies for the different domains related with the EOp logs (e.g., users, tools, etc.) provides a means for deciding log events are represented at some proper level of abstraction suitable for the analysis.

In particular we focus here on taxonomical information, such as that roughly sketched in Fig. 26, which provides a (partial) classification scheme for project tasks (i.e., the subject of log instances related to *task run* enterprise operations).

Such a hierarchy allow for choosing different abstraction levels for representing the instances of their associated domain, in a flexible and dynamic fashion.

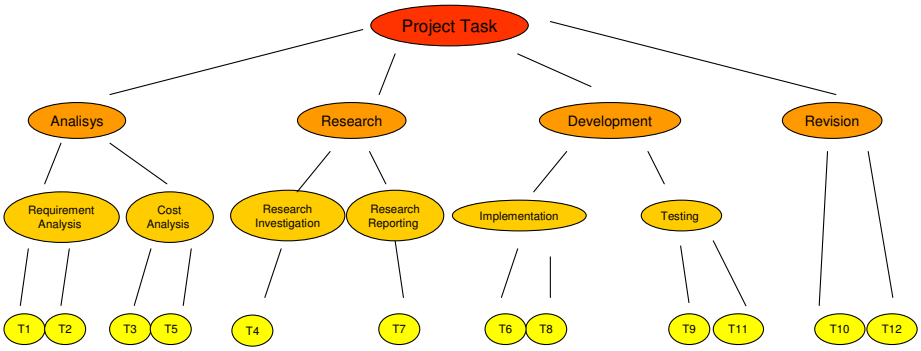

Fig. 26 A classification hierarchy over project tasks

The messages passing on ESB and logged on ESB server constitutes the knowledge base to extract process schema, each of one is constituted by a particular sequence of events happened in local tool. To obtain an more flexible and application of the process mining techniques, dynamic mechanisms has been provided to individuate the process execution instances for which a workflow schema is required. This way, process mining algorithms can be applied to a generic subset of operations registered in the knowledge base, so allowing an analysis of the dynamic context in which operations are executed. This enable to extend process mining to every procedural aspect of the architecture and not only to the ones that are explicitly referred to a process attribute of the logs.

The whole approach is detailed as follow:

A) EOp log restructuring

A.1) selection of a subset of EOp instances contained in the messages

A.2) mapping EOps on process activities

A.3) grouping of task execution in process instances

A.4) process log producing

B) Execution of a process mining algorithm on log so obtained

```
<?xml version="1.0" encoding="UTF-8" ?>
- <WorkflowLog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="WorkflowLog.xsd" description="Created by KMS+ ProM
  framework">
  <Source program="ATM" />
  - <Process id="project_task" description="">
    - <ProcessInstance id="KMS-plus" description="">
      - <AuditTrailEntry>
          <WorkflowModelElement>RequirementAnalysis;\n</WorkflowModelElement>
          <EventType>Start</EventType>
          <Timestamp>2007-01-01T14:00:00.000+01:00</Timestamp>
          <Originator>Frank</Originator>
        </AuditTrailEntry>
      - <AuditTrailEntry>
          <WorkflowModelElement>RequirementAnalysis;\n</WorkflowModelElement>
          <EventType>Complete</EventType>
          <Timestamp>2007-01-20T14:00:00.000+01:00</Timestamp>
          <Originator>Frank</Originator>
        </AuditTrailEntry>
      - <AuditTrailEntry>
          <WorkflowModelElement>CostAnalysis;\n</WorkflowModelElement>
          <EventType>Start</EventType>
          <Timestamp>2007-02-01T14:00:00.000+01:00</Timestamp>
          <Originator>Thomas</Originator>
        </AuditTrailEntry>
      - <AuditTrailEntry>
          <WorkflowModelElement>ResearchInvestigation;\n</WorkflowModelElement>
          <EventType>Start</EventType>
          <Timestamp>2007-02-02T14:00:00.000+01:00</Timestamp>
          <Originator>Lewis</Originator>
        </AuditTrailEntry>
      - <AuditTrailEntry>
          <WorkflowModelElement>CostAnalysis;\n</WorkflowModelElement>
          <EventType>Complete</EventType>
          <Timestamp>2007-03-31T14:00:00.000+02:00</Timestamp>
          <Originator>Thomas</Originator>
        </AuditTrailEntry>
      - <AuditTrailEntry>
          <WorkflowModelElement>ResearchInvestigation;\n</WorkflowModelElement>
          <EventType>Complete</EventType>
          <Timestamp>2007-04-02T14:00:00.000+02:00</Timestamp>
          <Originator>Lewis</Originator>
        </AuditTrailEntry>
      - <AuditTrailEntry>
          <WorkflowModelElement>ResearchReporting;\n</WorkflowModelElement>
          <EventType>Start</EventType>
          <Timestamp>2007-04-04T09:00:00.000+02:00</Timestamp>
          <Originator>Lewis</Originator>
        </AuditTrailEntry>
      - <AuditTrailEntry>
          <WorkflowModelElement>Implementation;\n</WorkflowModelElement>
          <EventType>Start</EventType>
          <Timestamp>2007-04-06T14:00:00.000+02:00</Timestamp>
          <Originator>franz</Originator>
        </AuditTrailEntry>
      - <AuditTrailEntry>
          <WorkflowModelElement>ResearchReporting;\n</WorkflowModelElement>
```


Fig. 27 Excerpt of a process log extracted

During the A.1 phase , to obtain a well defined subset of EOp instances, selection conditions can be specified with reference to different properties associated to the operations, as illustrated in Fig. 25: kind of operation, tool by which the operation is performed, user that enact the operation, project in which operation is involved, date of execution.

The A.2 phase allow to specify what kind of activities constitute the process thati is going to be examined, with respect to the attributes selected in the previous phase. This step specifies what level of abstraction will be adopted to represent the process. In particular, in the more detailed case, an activity of the process is referred to an operation executed by a certain user, on a certain object, using a certain tool. So every node in the workflow schema discovered by the process mining will be labelled by a n-upla:

<*eop, subject, tool, user*>

that represents one of the process activity.

To obtain a less detailed, and so more concise, representation, during the definition of the activity required for the process it is possible to abstract from some aspects related to the operation execution, i.e. if we consider just the tool on which the operation is performed, so disregarding other properties, it is possible to obtain a process model describing workflow among different tools.

The goal of the A.3 phase is to restructure events that are obtained by previous phase. Two different approaches are available:

- *Project-oriented*: every process instance is referred to the execution of a project
- *User-oriented*: every process instance is referred to the set of operations executed by a user, aggregated respect on a certain temporal period (daily, weekly...)

In particular, the project-oriented approach requires that all events are grouped in process instances respect on the project to which they are correlated. In other words, every process instance is constituted by a sequence of operations that are executing during a certain project. This kind of approach is most relevant in order to analyze goal-oriented context as enterprises.

User-oriented approach, instead, is based on a different way to re-organize the activities traced in process instances: every instance does not relate to a project execution but assembles the set of operation executed by a user. This approach allows to discover pattern of behaviour of the user, that are useful to understand the modus operandi of a user or a class of user. Some examples of analysis that should be supported are: how programmer users work, on a weekly period, on project classified as “research project”; how analyst users interact with tools on a daily period (the result of this analysis can be useful to elaborate a personalization of user interface).

During the A.4 phase, the process instances are transcribed in a process log consistent with respect to MXML format. Finally, during the B phase, process mining techniques extract a process model that express knowledge on worjflow characterizing the process analyzed.

5.5.2 Applying process mining to restructured logs

Fig. 28 illustrates the process log obtained applying the proposed approach to the EOp log showed in Fig. 25. In this example, only *task run* operations are considered and activities are mapped on *subject* of these operations (in the case of the task run operations, the subject is the task itself). They adopt a level of abstraction higher than the concrete task names that appear in the log, in particular, has been sorted out the second level in the hierarchy shown in Fig. 26 (containing, e.g., categories *RequirementAnalysis*, and *Revision*). The projects, instead, are considered as process instances, i.e., each process instance corresponds to the sequence of EOp instances that have been executed for a given project.

The application of process mining techniques on this log allows to extract knowledge about workflow characterizing the execution of different projects. Fig. 28 is a screenshot of the process mining tool integrated in our architecture, which illustrates the results obtained by using the algorithm in [97]. On the left side is reported a taxonomy of schemas and the root is shown on right side. In real and

complex scenario this kind of hierarchies allow the analysis of collaborative context and are adopted to build process ontologies.

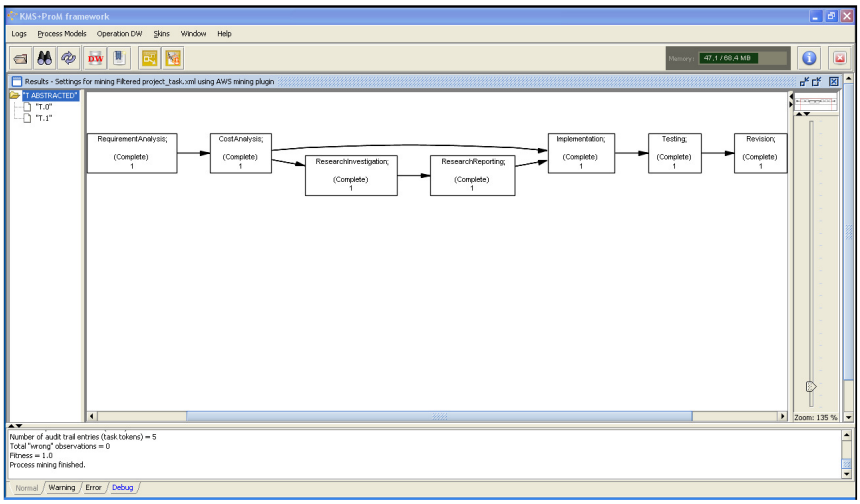

Fig. 28 Process model discovered for the log

6

Ontology for modelling business process knowledge

Summary: A novel approach to model processes and workflows is presented. It is based on the OntoDLP language, an extension of Disjunctive Logic Programming with object-oriented features. Compared to traditional models, the approach enables knowledge inference on dynamic structures of the process, thanks to the reasoning capabilities of OntoDLP. Moreover, the approach can be also used to redefine and classify existing workflow schemes. Indeed, their execution traces, produced by workflow engines, can be easily imported through the mapping facilities of the underlying metamodel, and eventually organized into taxonomic structures for modeling different execution-patterns.

6.1 A metamodel for process logic representation

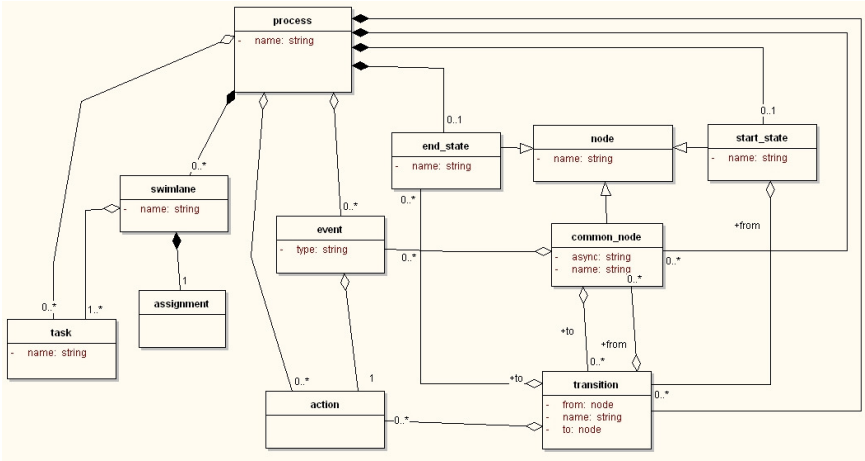

Fig. 29 A portion of process metamodel

The approach proposed in this chapter is based on a metamodel allowing for an intuitive graph-oriented representation of processes, based on the explicit definition of node and transition [102]. The metamodel includes a set of constructs allowing to abstract workflow solutions adopted by a large number of tools in the open source

community. We implement this metamodel by using the OntoDLP language, an extension of Disjunctive Logic Programming (DLP) with object-oriented features. We are able to define classes of processes and activities that can be both composed, to make process schemas, and classified to obtain hierarchical structures. Importantly, thanks to the reasoning capabilities provided by the DLP, we can define logic rules to analyze and automatically classify the traces of processes execution and to infer new knowledge about process-schema structures.

While the metamodel adopted in our approach is graph-oriented, it provides explicit constructs to express node and transition elements. It is derived from JPDL modeling approach just to allow an easy mapping to process traces generated by jBPM workflow engine.

As shown in Fig. 29, we consider a process as a composition of nodes, events, actions and task. A task should be associated both to a process or to specific portion of it. As in JPDL, we adopt a "swimlane" item to express a group of tasks that refer an unique assignment. Assignment shall be then associated to many kind of actors, based on particular workflow execution context. We connect "event" and "action" items to "node" and "transition" respectively, just to express that a transaction executed anywhere should be acknowledged in our context and associated to a specific state of our process schema. About nodes, we distinguish the initial and the ending point of the process, by using "start state" and "end state" respectively.

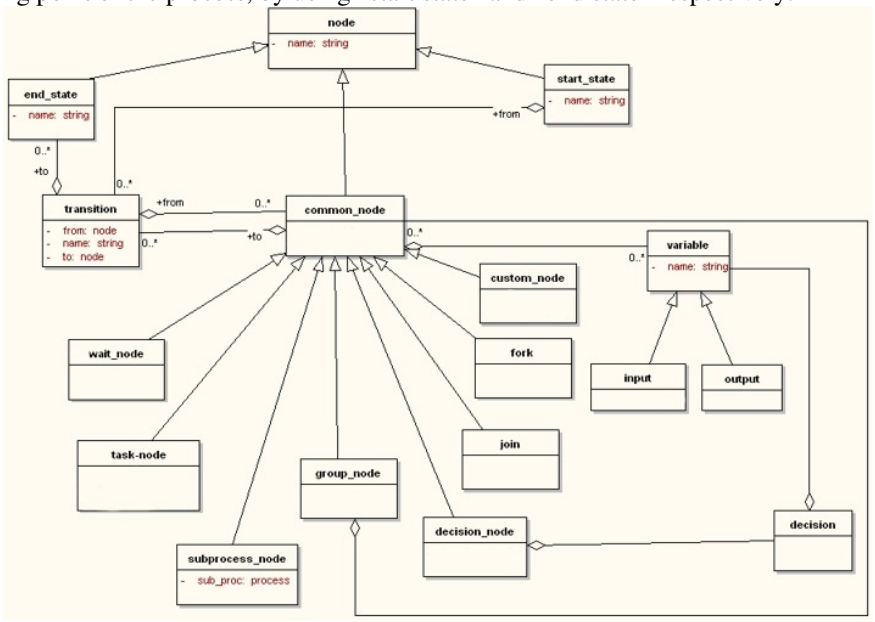

Fig. 30 A focus on node constructs

All other nodes are classified as "common nodes". As shown in Fig. 30, we assume many kind of common nodes. For example, "fork", "join" and "wait node" are flow control nodes. A "decision node" is a particular node which is associated to a "condition" and to an "handler" that is an entity able to resolve the issue. A task node, should instead be associated to one or more task assigned to an actor. Every common

node has also one or more variables that can be used to map input and output elements of the activity.

A particular kind of node is a “subprocess node” that is an activity that refers an external process to the current one. Significantly different is a “group node” that contains a set of activities without any constraint about their composition. By defining specific relations among a “group node” and many common nodes, we are able to express that a particular node is a collection of nodes. We can also use these collections to facilitate the categorization of processes. A group node, in fact, should express a common semantics that results an abstraction of the semantics connected to the single activities. So, for example, a “development” activity should be composed by an “implementation” and a “test” step.

6.2 Process representation and reasoning

Our approach is based on the formal representation of processes, according with the metamodel illustrated in the previous section, in the logic-based language OntoDLP introduced before: all constructs used for this process representation are defined in as previous chapter. On top of such a representation, we can then specify a number of (OntoDLP) inference rules, which allow us to discover new process properties and capture also dynamic knowledge which is hidden in process schemas.

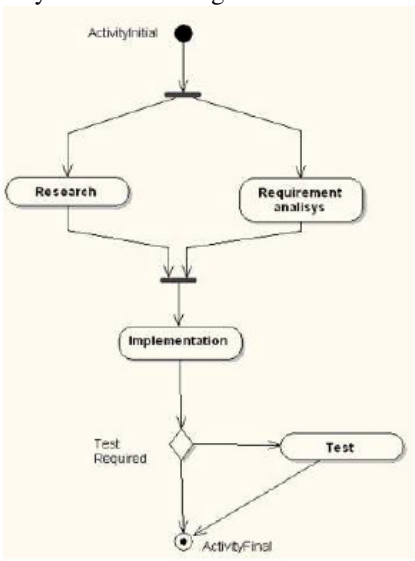

Fig. 31 An example of process schema

A process schema is a definition of a path of execution of activities that can be enacted many times. Every execution of a schema is a process instance in which variables have their value assignment. Before the instances generation, in the process schema are so defined only classes of activities that can admit different enactments in relation to different values assigned to their variables. We represent these classes of activities as specializations of the nodes introduced in the metamodel.

This way, a particular activity belonging to a specific process, is modeled as a specialization of one of the classes specifying an “activity node” element of the metamodel. So, for example, “requirement analisys”, “research”, “implementation” and “test” activities are subclasses of “task node” and may be involved in a “project” element, modeled as a “process” subclass:

```
class project isa {process}.

class research isa {task_node}.

class requirement_analisys isa {task_node}.

class implementation isa {task_node}.

class test isa {task_node}.

relation has_research_task(proj:project, res:research).

relation has_requirement_analysis_task(proj:project,
req_anal:requirement_analysis).

relation has_implementation_task(proj:project, impl:implementation).

relation has_test_task(proj:project, test:test).
```

We can then define a pattern of execution for a “project” type process, by declaring a set of subclasses of the “transition” element associating pairs of activities:

```
class start_fork_transition isa {transition} (from:start_state,
to:fork).

class research_join_transition isa {transition} (from:research,
to:join).

class requirement_analisys_join_transition isa {transition}
(from:requirement_analisys, to:join).

class decision_test_transition isa {transition} (from:decision_node,
to:test).

class test_end_state_transition isa {transition} (from:test,
to:end_state).
```

Moreover, similarly to the nodes, also these specific transitions must be associated to the specific ``project'' element.

In our approach, every instance of a process will generate several logical facts. So, for example, an instance of the above schema for a KMS process is here formalized as follows:

```
#7:process(name:"KMS").

#8:research(name:"KMS research", asynchronous:"true").

#9:requirement_analisys(name:"KMS requirement analisys",
asynchronous:"true").

#10:test(name:"KMS test", asynchronous:"true").

#11:join(name:"join node", asynchronous:"true").

#12:research_join_transition (name:"KMS research-join
transition",from:#8, to:#11)

#13:requirement_analisys_join_transition (name:"KMS requirement
analisys-join transition",from:#9, to:#11)
```

By performing reasoning on these facts we are able to infer new knowledge on static and dynamic aspect of processes. For example, we can define a rule expressing that every process that involves a “requirement analysis” and an “implementation” activity is “project” type process.

```
P:project(N):- contains(process:P, node:N1),
contains(process:P,node:N2), N1:requirement_analisys(),
N2:implementation(), P:process(name:N).
```

This way, the KMS process above defined, will be classified also as instance of “project” class. Adopting recursively this approach, we are able to recognize a “research and development project” as a project that involves also a “research” activity.

```
P:research_development_project(N):- contains(project:P, node:N1),
N1:research(),P:project(name:N).
```

where class research_development_project is defined as

```
class research_development_project isa {project}.
```

This way we are able to define a hierarchical structure of process schemas. When we start to design a process, we can use this hierarchical structure to find an appropriate schema for modeling a specific context. If we modify this schema, by adding or removing activities, we will be always able to automatically classify new instances, by using specific reasoning rules.

By example, if an ontology or a quality certification system provides a document classification we are able to classify a generic activity that receives as input a “notification” and produces as output a “research deliverable” as a research activity, using the following rule:

```
C:research(name:N, asynchronous:"true"):- C:node(name:N).

has_input(c_node:C, v:variable(name:"notification")),
has_output(c_node:C, v:variable(name:"research deliverable")).
```

where relations has_input and has_output are defined as follows:

```
relation has_input(c_node:common_node, v:variable).

relation has_output(c_node:common_node, v:variable).
```

Moreover, also if an activity is not modeled as atomic node, we can discover it in a path of activities that receives a specified input and produces a specified output. For example, we can define a project as a “research and development project” if it contains a research activity and a “development”, i.e. a path from a node that receives a “requirement analysis document” to a node that produces a “test report”

```
P:research_development_project(N):- P:process(name:N),

contains(process:P, node:N1),

contains(process:P, node:N2),

contains(process:P, node:N3),

N1:research(),

has_input(c_node:N2, v:variable(name:"requirement analysis
document")),

has_output(c_node:N3, v:variable(name:"test report")), path(from:N2,
to:N3).
```

This way, we are able to capture also dynamic knowledge that is hidden in process schemas.

6.3 Implementation and future works

The approach introduced in this paper has been implemented in OntoDLV system [7], that is an ontology management platform based on OntoDLP language and allowing to create, modify, navigate and query ontologies using a user-friendly visual environment. The metamodel adopted and presented in this work has been defined using the graphical interface and validated by the consistency check offered by the system. The addition of reasoning modules in OntoDLV allows the extraction of new knowledge about process schemas. In fact OntoDLV guarantees inference capabilities thanks to the integration of DLV system, widely recognised as the state of the art in the field of non monotonic reasoning (and disjunctive logic programming). For complexity analysis issues in OntoDLV refer to DLV results, shown in [8].

The long-term goal of this approach is to provide a support in the whole process management life-cycle. Actually, the metamodel has to be integrated in a framework for specifying enterprise models [6]. This way, it is possible to obtain an ontology of organizational processes that should support an architecture of heterogeneous open source tools for enterprise activities, like project planning and monitoring, timesheet compiling and analyzing, document management. Just to be easily mapped on JBoss process framework, widely adopted in open source community, the metamodel is inspired to JPDL formalism. With respect to JPDL and other xml-based languages, the proposed approach is able to use inference rules of DLP. This is particularly useful to link the triggers generated by generic JBoss-based tools to particular process events. Moreover, logic rules make possible to discover semantic dependencies inside process elements: actually, as it is illustrated in this paper, hierarchical structures are set just on the belonging of activities to process schemas; our purpose is to reason and to extract hierarchies also on the behaviour of processes.

The inference rules should be semi-automatically suggested by integrating process mining techniques that examine process instances. Process structures obtained should be useful in the process design phase: using OntoDLV querying we are able to find classes of process either composed by particular activities or associated to specific parameters or actors. A correlated future work regards the definition of techniques to semi-automatically compose a particular process schema as a function of the provided input, the required output and the existing classes of activities.

Conclusions

A core challenge in Business Process Management is the continuous, bi-directional translation between a business requirements view on the process space of an enterprise and the actual process space of this enterprise, constituted by the multiplicity of IT systems, resources, and human labour. However BPM does not provide a uniform representation of an organization's process space at a semantic level, which would be accessible to intelligent queries or for compliance checks. The advent of Semantic Business Process Management increased the level of automation of BPM by representing the various spheres of an enterprise using ontology languages and Semantic Web Services frameworks. Nevertheless no one complete methodology or framework have imposed as a standard in the SBPM that actually constitutes a universe of local solution and approaches. Moreover, existing Information Systems constitutes an heterogeneous scenario of solutions and a standardization is not a reasonable perspective, also because enterprise processes involved heterogeneous functional areas.

Therefore, the evolution towards SBPM have to be progressive and modular. This thesis propose a methodology for a continuous planning, analysis and enhancement of process-oriented IS oriented to capture and support an update of the system based on the observation of the users' behaviour. On another side, a set of semantic solution to support the whole lifecycle of the BPM are introduced. Process mining algorithms seem actually to be the most concrete basis to provide machine-accessible representations of the BPM scenario. However, until now, Process mining has been focused only on control flow and workflow. A few attention has been reserved to the data and organization domains connected to the process. In other words, raditional process mining techniques addressed the discovery of workflow models (the so called process-mining *control-flow* perspective) by focusing on the occurrences of process tasks in the registered logs, thereby completely disregarding any other kind of information usually kept by real systems, such as activity executors, time-stamps, parameter values, and various performance data. In actual fact, recently, some efforts have been spent in the problem of extracting knowledge on social collaborations from process logs (the so-called *organizational* process-mining perspective). In [107] an approach to obtain a structural network based on users interactions is obtained from process analysis. In [13], instead, recent innovations consider an ontology to represent the domain connected to the process execution, but this ontology is cabled inside the

process definition code. The solutions that have been introduced in our thesis are instead based on a close relationship between the description of process and its domain, in which ontologies support process analysis and process analysis enables the ontology evolution and the domain knowledge discovering.

This relationship is one of the most relevant enhancement introduced by the solution introduced in chapter 5 to support loosely-structured collaborative process. In fact it enables the discovering of new process schema with respect to task that are obtained by an ontology-based restructuring of operations log. In this approach, the tasks that are related to process steps can be a conceptualization of project unit as well as document or user. This way, after the process mining analysis, we obtain taxonomic structures in which every level represents a description of classes of task with a different level of abstraction on attributes and so we are able to offer a classification of project unit as well as other domain elements. This approach has actually ultimate a first experimentation and requires now more detailed enhancements on different aspects:

- the enterprise model has to be extended to offer a logic-oriented representation of dynamic aspect of the behaviour
- the ontology-based knowledge extraction in the operation logs has to consider a portion of operation domain wider than the subject description provided in this phase: by this point of view a semantic enrichment of the operation description will be useful
- the output schemas have to be automatically mapped in the logic formalism adopted for the reasoning on the knowledge, to obtain an integrated discovery process

In particular, the framework that we adopt for enterprise knowledge representation is actually under development [105, 106] to implement a (semi) automatic mechanism for enterprise entity annotation respect on an ontological topic. Actually this aspect is focused especially on a semantic approach for knowledge extraction in structured and unstructured document. In a following phase, the power of the Disjunctive Logic Programming used to implement the approach will be able to allow to throw new knowledge about domain knowledge also on the process knowledge base.

References

[1] S. Staab, D. O'Leary (eds.) (2000) Bringing Knowledge to Business Processes, Proceedings of 2000 AAAI Spring Symposium, AAAI Press, 2000

[2] E. Fillion; C. Menzel, T. Blinn, R. Mayer (1995), An Ontology-Based Environment for Enterprise Model Integration. Paper presented at the IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing, 19-20 August1995, Montreal, Quebec, Canada.

[3] M.S. Fox (1992), "The TOVE Project: Towards A Common-sense Model of the Enterprise", Enterprise Integration Laboratory Technical Report

[4] M.S. Fox, M. Gruninger (1998), "Enterprise Modelling", AI Magazine, AAAI Press, Fall, pp. 109-121.

[5] B. Heuluy, F.B. Vernadat (1997), The CIMOSA Enterprise Ontology, Proceedings of the IFAC Workshop-MIM'97, Vienna, February 3-5,1997.

[6] Gualtieri A., Ruffolo M. (2005): An Ontology-Based Framework for Representing Organizational Knowledge, Proceeding of I-Know '05 - 5 th International Conference on Knowledge Management, Graz Austria

[7] OntoDLV system, http://www.exeura.it/ontodlv

[8] Ricca F., Leone N. (2006): Disjunctive Logic Programming with Types and Objects: The DLV+ System. Journal of Applied Logics Elsevier ISSN: 1570-8683 (to appear); KBS Research Reports INFSYS RR-1843-05-10 Institut für Informationssysteme Technische Universität Wien Favoritenstrasse 11 A-1040 Vienna Austria

[9] Booch G., Rumbaugh J., and Jacobson I. (1998), The Unified Modeling Language User Guide, Addison-Wesley.

[10] Center for Technology in Government, University at Albany, A Survey of System Development Process Models CTG.MFA – 003, Technical Report, available at http://demo.ctg.albany.edu/publications/reports/survey_of_sysdev/survey_of_sysdev.pdf

[11] Coad, P., De Luca, J., and Lefebre, E. (1999), Java Modeling in Color with UML: Enterprise Components and Process, Prentice Hall.

[12] Cockburn, A. (2002), Agile Software Development, Addison-Wesley.

[13] Department of Technology Management, Eindhoven Technical University, The ProM Framework, available at http://is.tm.tue.nl/~cgunther/dev/prom/

[14] Eilam, E., Reversing (2005): Secrets of Reverse Engineering, Wiley.

[15] Exeura - Knowledge Management Solutions, http://www.exeura.it

[16] Fensel, D., Ontologies (2001): A Silver Bullet for Knowledge Management and Electronic Commerce, Springer Verlag.

[17] Greco G., Guzzo A., Manco G., and Saccà D. (2005), "Mining and Reasoning on Workflows", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 4, 2005, pp. 519-534.

[18] Royce, W.W. (1970), "Managing the Development of Large Software Systems", Proc. of IEEE WESCON, pp. 1-9.

[19] Wetzstein B.; Ma Z.; Filipowska A.; Kaczmarek M.; Bhiri S.; Losada S.; Lopez-Cobo, J.M.; Cicurel L. (2007): Semantic Business Process Management: A Lifecycle Based Requirements Analysis. Proceedings INPROC-2007

[20] Leymann, Frank; Roller, Dieter (2000): Production Workflow - Concepts and Techniques. PTR Prentice Hall, 2000.

[21] van der Aalst, W.M.P; van Dongen, B.F.; Herbst, J.; Maruster, L.; Schimm, G.; Weijters, A.J.M.M (2003): Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering 47 (2003) 237–267.

[22] Smith, Howard; Fingar, Peter: Business Process Management. The Third Wave. Meghan-Kiffer,US 2003.

[23] Hepp M.; Leymann F.; Domingue J.; Wahler A.; Fensel D. (2005): Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. Proceedings of the IEEE ICEBE 2005, October 18-20, Beijing, China, pp. 535-540

[24] Hepp M; Roman D (2007): An Ontology Framework for Semantic Business Process Management, Proceedings of Wirtschaftsinformatik 2007, February 28 - March 2, 2007, Karlsruhe.

[25] Irene Celino, Ana Karla Alves de Medeiros, Gernot Zeissler, Michael Oppitz, Federico Facca, Stefan Zoeller: Semantic Business Process Analysis (2007), Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM-2007), Vol-251, CEUR-WS, June 2007, ISSN 1613-0073

[26] van der Aalst, W.M.P.; Pesic, M. (2006): Specifying, Discovering, and Monitoring Service Flows: Making Web Services Process-Aware. BPM Center Technical Report, No. BPM-06-09, 2006.

[27] Fox, Marc S. et al. (1998): An Organisation Ontology for Enterprise Modeling. In: M. Prietula; K. Carley; L. Gasser (Hrsg.): Simulating Organizations: Computational Models of Institutions and Groups. AAAI/MIT Press, Menlo Park CA 1998, pp. 131-152.

[28] Gruninger, Michael et al. (2000): Ontologies to Support Process Integration in Enterprise Engineering. In: Computational & Mathematical Organization Theory 6 (2000) 4, pp. 381-394.

[29] Dietz, Jan L. G. (2006): Enterprise Ontology. Springer, Berlin / Heidelberg

[30] I. Weber, J. Hoffmann, J. Mendling, J. Nitzsche (2007), Towards a Methodology for Semantic Business Process Modeling and Configuration, Proceedings of the 2nd International SeMSoC Workshop ``Business Oriented Aspects concerning Semantics and Methodologies in Service-oriented Computing" (SemSoc'07), at ICSOC'07, Vienna, Austria

[31] Davenport, T.H., and Short, J.E. (1990): The New Industrial Engineering: Information Technology and Business Process Redesign. In Sloan Management Review, Vol. 31, No. 4, pp. 11-27.

[32] zur Muehlen, M., 2001. Process-Driven Management Information Systems - Combining Data Warehouses and Workflow Technology. In Proc. of the 4th ICECR-4 Int. Conf., pp. 550-566.

[33] Serrano, A. 2003. Capturing Information System's Requirement Using Business Process Simulation. In Proc. of the 15th ESS Int. Conf.

[34] Kim, K-H, Yoo, H.-J., and Kim, H.-S., 2005. A Process-Driven E-Learning Content Organization Model. In Proc. of 4th IEEE ACIS Int. Conf., pp. 328-333.

[35] Grover, V., Fielder, K.D., and Teng, J.T.C., 1994. Exploring the Success of Information Technology Enabled Business Process Reengineering. In IEEE Transactions on Engineering Management, Vol. 41, No. 3, pp. 276-284.

[36] van Meel, J.W., Bots, P.W.G., and Sol, H.G., 1994. Towards a Research Framework for Business Engineering. In IFIP Transactions A: Computer Science and Technology, Vol. 54, pp. 581-592.

[37] Teufel, S., and Teufel, B., 1995. Bridging Information Technology and Business: Some Modeling Aspects. In SIGOIS Bulletin, Vol. 16, No. 1, pp. 13-17.

[38] Earl, M.J., 1994. The New and the Old of Business Process Redesign. In Journal of Strategic Information Systems, Vol. 3, No. 1, pp. 5-22.

[39] MacArthur, P.J., Crosslin, R.L, and Warren, J.R., 1994. A Strategy for Evaluating Alternative Information System Designs for Business Process Reengineering. In International Journal of Information Management, Vol. 14, No. 4, pp. 237-251.

[40] Giaglis, G.M., 2001. A Taxonomy of Business Process Modeling and Information Systems Modeling Techniques. In International Journal of Flexible Manufacturing Systems, Vol. 13, No. 2, pp. 209-228.

[41] Curtis, W., Kellner, M.I., and Over, J., 1992. Process Modeling. In Communications of the ACM, Vol. 35, No. 9, pp. 75-90.

[42] Vasconcelos, A., Caetano, A., Neves, J., Sinogas, P., Mendes, R., and Tribolet, J.M., 2001. A Framework for Modeling Strategy, Business Processes and Information Systems. In Proc. of the 5th IEEE EDOC Int. Conf., pp. 69-80.

[43] Castela, N., Tribolet, J.M., Silva, A., and Guerra, A., 2001. Business Process Modeling with UML. In Proc. of the 3rd ICEIS Int. Conf., Vol. 2, pp. 679-685.

[44] Sinogas, P., Vasconcelos, A., Caetano, A., Neves, J., Mendes, R., and Tribolet, J.M., 2001. Business Processes Extensions to UML Profile for Business Modeling. In Proc. of the 3rd ICEIS Int. Conf., Vol. 2, pp. 673-678.

[45] Neves, J., Vasconcelos, A., Caetano, A., Sinogas, P., Mendes, R., and Tribolet, J.M.. Unified Resource Modelling: Integrating Knowledge into Business Processes. In Proc. of the 3rd ICEIS Int. Conf., Vol. 2, pp. 898-904.

[46] Mendes, R., Mateus, J., Silva, E., and Tribolet, J.M., 2003. Applying Business Process Modeling to Organizational Change. In Proc. of the 2003 AMCIS Int. Conf.

[47] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana (2001) "Web Services description Language (WSDL) 1.1.," W3C, Note 2001.

[48] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. Ferguson (2005): Web Services Platform Architecture: Prentice Hall.

[49] Shapiro, R.(2002): A Comparison of XPDL, BPML and BPEL4WS. Cape Visions, http://xml.coverpages.org/Shapiro-XPDL.pdf

[50] Workflow Management Coalition (1999): Terminology and Glossary, Issue 3.0. Document Number WfMC TC-1011.

[51] Arkin A. (2002): Business Process Modeling Language, BPMI.org

[52] IBM (2002): Business process execution language web services, version 1.0

[53] van der Aalst W.M.P. (1998): The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, 8(1):21-66.

[54] Jboss (2005): jBPM Process De¯nition Language, version 3.0

[55] Fensel, D., Bussler, C. (2002): The Web Service Modeling Framework WSMF. Eletronic Commerce: Research and Applications. Vol. 1-2002. 113-137

[56] Fensel, D. and Motta, E. (2001): Structured Development of Problem Solving Methods. IEEE Transactions on Knowledge and Data Engineering, Vol. 13(6)-2001. 913-932.

[57] Casati F., Ceri S., Pernici B., and Pozzi G. (1995): Conceptual Modeling of Workows. In Proc. 14th Object-Oriented and Entity-Relationship Modelling , Gold Coast, Australia, December.

[58] Kappel G., Lang P., Rausch-Schott S., and Retschitzegger W. (1995): Workflow Management Based on Objects, Rules, and Roles. Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society, 18(1), pages 11{18.

[59] Kradolfer M.: A Workflow Metamodel Supporting Dynamic, Reuse-Based Model Evolution, University of Zrich, Ph. D. Thesis

[60] SWWS Consortium (2003). Report on Development of Web Service Discovery Framework. October 2003. http://swws.semanticweb.org/public_doc/D3.1.pdf

[61] Cabral, L. and Domingue, J. and Motta, E. and Payne, T. and Hakimpour, F. (2004) Approaches to Semantic Web Services: An Overview and Comparisons. In Bussler, C. and Davies, J. and Fensel, D. and Studer, R., Eds. Proceedings First European Semantic Web Symposium (ESWS2004)

[62] Kilov, Haim (1999), Business Specifications, Prentice Hall.

[63] Greco S., Leone N. and Rullo P. (1992): COMPLEX: An Object-Oriented Logic Programming System, IEEETKDE vol 4-1992.

[64] Baral C. and Gelfond M. (1994): Logic Programming and Knowledge Representation, JLP vol 19/20, 73/148-1994.

[65] Lobo J., Minker J. and Rajasekar A. (1992): Foundations of Disjunctive Logic Programming, The MIT Press, Cambridge, Massachusetts.

[66] Disjunctive Logic Programming and Disjunctive Databases, 13*th* IFIP World Computer Congress, Hamburg, Germany (1994).

[67] Eiter T., Faber W., Gottlob G., Koch C., Leone N., Mateis C., Pfeifer G. and Scarcello F. (1999): The DLV System, Workshop on Logic-Based Arti¯cial Intelligence, Washington, DC.

[68] Gelfond M. and Lifschitz V. (1991): Classical Negation in Logic Programs and Disjunctive Databases, NGC vol 9, 365{385.

[69] Lifschitz V. (1996): Foundations of Logic Programming, Principles of Knowledge Representation, 69{127.

[70] Minker J. (1994):Overview of Disjunctive Logic Programming, AMAI vol 12 1{24.

[71] Baral C. (2002): Knowledge Representation, Reasoning and Declarative Problem Solving, Cambridge University Press.

[72] M. Polanyi (1996): The Tacit Dimension, Routledge and Kegan Paul.

[73] M. Polanyi (1997): Tacit Knowledge. Chapter 7 in Knowledge in Organizations, Laurence Prusak, Editor. Butterworth-Heinemann, Boston.

[74] I. Nonaka (1994): A Dynamic Theory of Organizational Knowledge Creation. In Organization Science, 5-1994

[75] I. Nonaka, H. Takeuchi (1995): The Knowledge-Creating Company. Oxford University Press

[76] Hayes, P. J.(1985): The Second Naive Physics Manifesto, in Hobbs and Moore (eds.), Formal Theories of the Common-Sense World, Norwood: Ablex.

[77] McCarthy, J. (1980): Circumscription -- A Form of Non-Monotonic Reasoning, Artificial Intelligence, 5(13): 27-39, 1980.

[78] Fabien Gandon (2002) Ontology engineering: a survey and a return on experience. Technical report.

[79] N. Guarino and P. Giaretta (1995):. Ontologies and knowledge bases: Towards a terminological clarification. In N.J. Mars, editor, Towards Very Large Knowledge Bases –Knowledge Building and Knowledge Sharing, pages 25–32, Amsterdam, 1995. IOS Press.

[80] T. R. Gruber (1993): A translation approach to portable ontology specifications. Knowledge Acquisition, 5(2):199–220, 1993.

[81] Pim Borst. Construction of Engineering Ontologies for Knowledge Sharing and Reuse. PhD thesis, Tweente University, 1997.

[82] N. Guarino (1998): Formal Ontology and Information Systems, "Formal Ontology in Information Systems: proceedings of FOIS'98", N. Guarino (ed), IOS Press, p 3-15.

[83] O. Corcho and A. Gomez-Perez (2000). *A Roadmap to Ontology Specification Languages*. In R. Dieng and O. Corby, editors, 12th International Conference on Knowledge Acquisition, Modeling and Management (EKAW), pages 80--96, Juan-les-Pins, France, October 2-6, 2000. Springer.

[84] Farquhar, A., Fikes, R., Rice, J. (1996) The Ontolingua Server: A Tool for Collaborative Ontology Construction. Proceedings of KAW96. Banff, Canada.

[85] Genesereth, M., Fikes, R. (1992) Knowledge Interchange Format. Technical Report. Computer Science Department. Stanford University. Logic-92-1.1992.

[86] Kifer, M., Lausen, G., Wu, J. (1995) Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of the ACM.

[87] MacGregor, R. (1991) *Inside the LOOM clasifier*. SIGART bulletin. #2(3):70-76. June, 1991.

[88] Finin, T., et al. (2003) Automatically generated DAML markup for semistructured documents'. Proceedings of the 2003 AAAI Spring Symposium on Agent-Mediated Knowledge Management (AMKM).

[89] Fensel, D., et al (2001) OIL: An ontology infrastructure for the semantic web',. *IEEE Intelligent Systems*, March/April 2001.

[90] Horrocks, I. & Harmelen, F.V. (2001) Reference Description of the DAML+OIL Ontology Markup Language.

[91] Luke, S. & Heflin, J. (2000) SHOE 1.01 Proposed specification. SHOE Project.

[92] Karp, R., Chaudhri, V., Thomere, J. (1999) XOL: An XML-Based Ontology Exchange Language.

[93] Bray, T., Paoli, J., Sperberg, C. (1998) Extensible Markup Language (XML) 1.0. W3C Recommendation. http://www.w3.org/TR/REC-xml.

[94] Lassila, O., Swick, R. (1999) Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation. http://www.w3.org/TR/PR-rdf-syntax.

[95] Brickley, D., Guha, R.V. (1999) Resource Description Framework (RDF) Schema Specification. W3C Proposed Recommendation. http://www.w3.org/TR/PR-rdf-schema.

[96] Greco G, Guzzo A, Pontieri L. (2005) Mining hierarchies of models: from abstract views to concrete specifications. In Proc of the 3rd International Conference on Business Process Management (BPM'05), pages 32--47.

[97] Greco G, Guzzo A, Pontieri L, Saccà D (2006) Discovering expressive process models by clustering log traces. IEEE Trans. on Knowledge and Data Engineering, 18(8): 1010--1027.

[98] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst (2005). The ProM framework: A new era in process mining tool support. In Proc. of 26th Intl. Conf. on Applications and Theory of Petri Nets (ICATPN'05), pages 444-454, 2005.

[99] Dean M., Connolly D., van Harmelen F., Hendler J., Horrocks I., McGuinness D. L., Patel-Schneider P.F., and Stein L.A. (2003). OWL web ontology language reference. W3C Working Draft, 31 March 2003. Available at http://www.w3.org/TR/2003/WD-owl-ref-20030331.

[100] Dell'Armi T., Gallucci L., Leone N., Ricca F., and Schindlauer R.. ONTODLV: an ASP-based System for Enterprise Ontologies. In Proceedings ASP 2007: Answer Set Programming: Advances in Theory and Implementation, Porto, Portugal, pages 99-113, September 2007.

[101] A. Cuzzocrea, A. D'Atri, A. Gualtieri, A. Motro, D. Saccà (2007): Grid-VirtuE: A Layered Architecture for Grid Virtual Enterprises. Proceedings of Confenis07, IFIP International Conference on Research and Practical Issues of Enterprise Information Systems, Beijing, China

[102] A. Gualtieri, T. Dell'Armi, N. Leone (2006): Process representation and reasoning using a logic formalism with object-oriented features. Proceeding of BPI - Workshop on Business Process Intelligence (BPI) in conjunction with BPM 2006 Vienna, Austria

[103] A. Gualtieri, D. Saccà (2006): A process-driven modelling of information systems. Proceeding of Itais '06 – 3rd Italian Conference of Information Systems, Milano, Italy

[104] F. Folino, G.Greco, A.Gualtieri, A.Guzzo, L.Pontieri (2007): knowledge discovery and classification of cooperation processes for internetworked enterprises. Proceeding of Itais '07 – 4th Italian Conference of Information Systems, Milano, Italy

[105] M. Ruffolo, M. Manna (2006). "A Logic-Based Approach to Semantic Information Extraction". Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS'06), Paphos, Cyprus, May 23-27, 2006

[106] M. Ruffolo, N. Leone, M. Manna, D. Saccà (2006). "Towards a Semantic Information Extraction Approach from Unstructured Documents". Proceedings of the fourteenth Italian Symposium on advanced Database System (SEBD'06), Ancona, Italy, June 18-21 2006

[107] J. Scott. Social Network Analysis. Sage, Newbury Park CA.