Doctoral School on Engineering Sciences
Università Politecnica delle Marche

Extended summary

Formal Methods for Semantic Case Management

Curriculum in Ingegneria Informatica, Gestionale e dell'Automazione

Author

Lorenzo Boaro

Tutor

Prof. Luca Spalazzi

Date: 30-01-2013

Abstract. During the years *Business Process Management* (BPM) has emerged as the go-to approach for handling *routine work*. As a result, business processes can be structured and therefore automated since characterized by a certain level of repeatability. *Case Management* (CM) and, more recently, *Adaptive Case Management* (ACM) were coined for more dynamic approaches taking into account *knowledge work*. Unlike the first, CM and ACM fit in managing processes characterized by a certain degree of uncertainty. Even if the principles of scientific management propose a clear distinction between routine and knowledge work, organizations cannot do the same. Almost all work that people do consists of routine work and non-routine work.

Based on previous considerations we propose a theoretical systematization of the aspects involved to represent *knowledge-intensive processes*. The overall goal is to integrate BPM and CM aspects and formalize, through behavioral and semantic technologies, each element taking part in the design and the execution of business processes. Each process is described through a new modeling language. Starting from a model, we provide its operational semantics. The result consists into parametric and annotated state-based systems. Formal methods based on *Model Checking* techniques can be applied to introduce verification and selection functionalities both at *design-time* and *run-time*.

Keywords. business process management, case management, formal methods, semantic annotations, modeling language

1 Problem statement and objectives

As a methodology, *Business Process Management* (BPM) involves a large effort to identify the structure of business processes which can be instantiated and executed through traditional process automation. For *routine work* this approach works well. Routine work is well known and can be planned to some level of details. Due to its repeatability, processes are *structured* and, thus, can be automated [1].
Nowadays organizations are dealing with trends that increase the complexity of their business. Routine work is less and less common for many organizations. Its opposite is called *knowledge work* and does not have the same level of predictability. Knowledge work introduces *unstructured* processes. Such processes are called *knowledge-intensive processes* since there is no predefined view of the process and tasks are discovered during their execution [2]. Obviously it is not possible to make a clear distinction between routine and knowledge work. Almost all work that people do consists of predictable phases mixed together with non-routine ones. In this scenario, *Case Management* (CM) and, more recently, *Adaptive Case Management* (ACM) were coined for more dynamic and rule-oriented approaches.
While BPM has received broad attention in the scientific community and numerous formalisms have been provided to describe structured processes (see [3], [4], [5], [6]), engineering of knowledge-intensive processes is far from being mastered. Only few approaches have been taken into consideration (see [7], [8], [9]). Providing formalisms for describing processes characterized by an unpredictable nature can lead to different advantages. In particular, formal methods can be applied to support workers during their tasks. Benefits can be obtained both during the modeling of the process (*design-time* phase) and its execution (*run-time* phase). Side effects consist in enabling efficient and quality-improved processes in environments where work cannot be determined a priori.

Continuing to be mere repositories of information, systems do not have capabilities to interpret information and do not allow people to get the data they need to work and to reason. In the overall scenario current problems consist of terminology mismatch and unstructured and isolated knowledge representation processes. Semantic aspects can be used to specify processes more precisely. Furthermore, interoperability problems can arise. In environments where collaboration among different actors is required, disambiguation of terms becomes a main aspect, specially whereas a huge space is present and schema for documenting model or terms are absent.

The aim of this work is to provide a theoretical framework for describing structured and unstructured processes. Such processes cannot be completely determined in advance since characterized on both routine and knowledge work. In particular, we focus on the possibility to model and execute what we have called *hybrid processes*. Users are free to perform activities as they prefer, but, at the same time, they can automate tasks when the latter are characterized by a predefined flow. Furthermore, *semantic annotations* permit to specify model elements in a more accurate manner and improve the collaboration among different actors.

2 Research planning and activities

Modeling can lead to advantages to organizations but, in general, work cannot be exactly identified. Routine work and knowledge work cannot be completely separated and, hence, processes can vary from a predefined structure to an undefined one. In addition, users are not simple performers of some types of tasks. They are *knowledge workers* [10] and, as such, they operate through expertise.

Based on previous considerations, we define a new modeling language, based on *Business Process Modeling Notation* (BPMN) [11], as the starting point to create a model of the "to-be" process, i.e. a *case*. Starting from a model, we translate the case into a parametric state-based system. Such a formalism makes possible to verify correctness properties of systems. Furthermore, cases are enriched by semantic annotations. Using *ontologies*, it is possible to describe elements and improve the collaboration.

The need to introduce formal methods has two main objectives. On the one hand, the introduction of an operational semantics allows to describe in a detailed manner the behavior of the elements that belong to the modeling notation - this in order to run cases in ad hoc execution engines. On the other hand, the formal development of a correct algorithm based on *Model Checking* techniques allows to apply verification and selection mechanisms - this in order to support users during their tasks.

2.1 Theoretical Background

The goal of this section is to provide a theoretical background that describes the basis of our work.

2.1.1 Ontology Representation

An ontology can be represented by means of *Description Logic* (DL) [12], a family of formal languages. The main operators are defined below.

Definition 1 Let A be a *concept name*. Let R be a *role name*. Let a be an *individual name*. Let C be a *concept*. Then:

- A *Concept* is defined as follows: $C ::= A \mid \top \mid \bot \mid C \sqcap C \mid C \sqcup C \mid \forall R.C$
- A *tbox* (*Terminology*) *Statement* is defined as follows: $\rho ::= C \sqsubseteq C$
- An *abox* (*Assertional*) *Statement* (or *assertion*) is defined as follows: $p ::= a : C \mid a.\mathrm{R} = a$

Each DL language is composed of the following symbols: *concept names*, *role names*, and *individual names* (or *individuals*). It also includes a set of constructors that permit the formation of *concepts*. A concept-based *knowledge base* is formed by two components: *tbox* (a set of concepts) and *abox* (a set of facts). While the former describes the ontology, the latter may represent the data of a knowledge base. We also introduce the notion of *query* as follows.

Definition 2 Let C be a *concept*. Let R be a *role name*. Let a be an *individual*. Let x (and y) be a *variable*. Then:

- $x : C$, $x.\mathrm{R} = a$, *or* $a.\mathrm{R} = x$ (resp. $x.\mathrm{R} = y$)

It denotes all the individuals such that, substituted for x (resp., all the pairs of individuals such that substituted for (x,y)), they make true the corresponding assertions.

2.1.2 Case Representation

Let us introduce the notion of *annotated case system* extending the definitions provided in [13] and [14].

Definition 3 An *annotated case system* U over a *Description Logic* language L is defined as $\langle \{\Sigma^A_i \mid i \in I\}, T, A_T \rangle$ where:

- Σ^A_i is a finite set of parametric and annotated transition system definitions;
- T is the terminology (*tbox*) of the annotation;
- A_T is the set of all the *concept* and *role assertions* defined over T.

An annotated case system is composed by a set of *annotated case* definitions.

Definition 4 An *annotated case* Σ^A_i defined over a *Description Logic* language L is a tuple $\langle n, S_i, S_i^0, G_i, R_i, \Lambda_i \rangle$ where:

- S_i is the finite set of states of Σ^A_i;
- $S_i^0 \subseteq S$ is the set of initial states of Σ^A_i;
- $G_i = \{g_i, ...\}$ is a set of guards used to label transitions of Σ^A_i;
- $R_i \subseteq S_i \times G_i \times S_i$ is the transition relation;
- $\Lambda_i : S_i \rightarrow 2^{A_T}$ is the annotation function.

Intuitively, a guard can be an assertion or a conjunctive query. For each state, the Λ function enumerates all its assertions.

2.1.3 *Specification Representation*

Verification requirements enable to express conditions on annotated case systems. Extending the work presented in [13], correctness properties are expressed using *Annotated Indexed-*CTL∗\X.

Definition 5 Let A_T a set of assertions. Let p a concept or role assertion. Let CQ be a set of conjunctive queries over T. Let q a conjunctive query. Let I be a set of possible indexes. Then, an *Annotated Indexed-*CTL∗\X formula can be defined as:

- $\varphi(i) ::= p(i) \mid q(i)[x] \mid \varphi(i) \wedge \varphi(i) \mid \neg\varphi(i) \mid \varphi(i)\ U\ \varphi(i)$
- $\Phi(i) ::= A\varphi(i) \mid E\varphi(i) \mid \bigwedge_i A\varphi(i) \mid \bigwedge_i E\varphi(i)$

Intuitively, $\varphi(i)$ defines the set of so-called state-formulas while $\Phi(i)$ constitute the set of path-formulas. Both contain at least an atomic proposition indexed by i. In the following, formulas without path quantifiers (A and E) form *Indexed-*LTL\X, a subset of *Annotated Indexed-*CTL∗\X.

2.1.4 *Verification Algorithm*

Model Checking can be undecidable due to both semantic annotations and parametric aspects. In a specification, annotations may denote a infinite set of individuals. On the other hand, automated methods to permit verification for arbitrary size systems it is undecidable in general. As a consequence, the algorithm, under the *closed world assumption*, first performs a procedure where annotations are "lowered" to a purely syntactic form. Then, using results presented in [13], it reduces Model Checking problem for systems for a small *cutoff* size *c*.

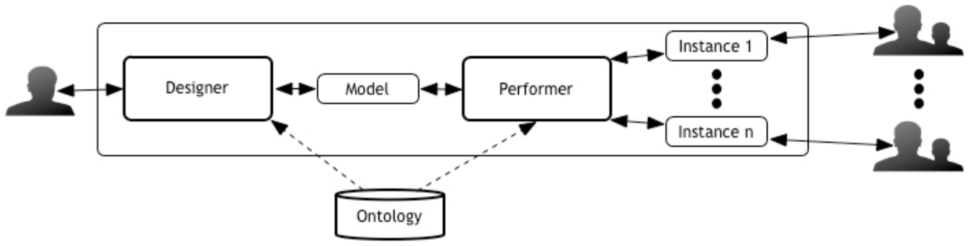

Figure 1. Framework overview

The main elements are described below:

- *Temporal specification normalization*: it translates the temporal specification into its *positive normal form*.
- *Relation building*: it considers all the queries, both in the temporal specification and in the guards of an annotated case system. Two different tables composed by positive and negative assertions are obtained.
- *Process grounding*: it substitutes the guards with disjunctions (or conjunctions) of states (that belong to other instances) where those guards hold. For each valid assignment, a different parametric state transition system is obtained.
- *Temporal specification grounding*: it substitutes the conjunctive queries that belong to the temporal specification with disjunctions of states where the queries hold. For each valid assignment, a different specification is obtained.
- *Propositional model checking*: it applies the propositional temporal logic model checking algorithm to a finite number of relatively small systems to which cutoff has been applied.

2.2 Tools

This section is devoted to provide an overview of the tools our theoretical framework makes available.

2.2.1 Framework

As depicted in Figure 1, it is possible to distinguish two main components: the *designer* and the *performer*. The former is used by the user who creates the model of the "to-be" process. The latter, instead, runs when the model is deployed and, hence, executed. They rely on a shared ontology which is adopted both in the design and the execution of a case process.

2.2.2 Modeling Language

The design phase is achieved by means of a new modeling language which captures the static aspects of a case process. As illustrated in Figure 2, the language provides few graphical elements that can be enriched through semantic annotations. In particular, it is possible to distinguish between nodes and connectors. A node can be an *Event*, an *Activity*, a *Case* or an *Artifact*. An event describes something that happens during the execution. An activity represents an atomic task. A case acts as a container of node definitions. Finally, an artifact defines data objects. Nodes can be linked together through connectors. A connector can be

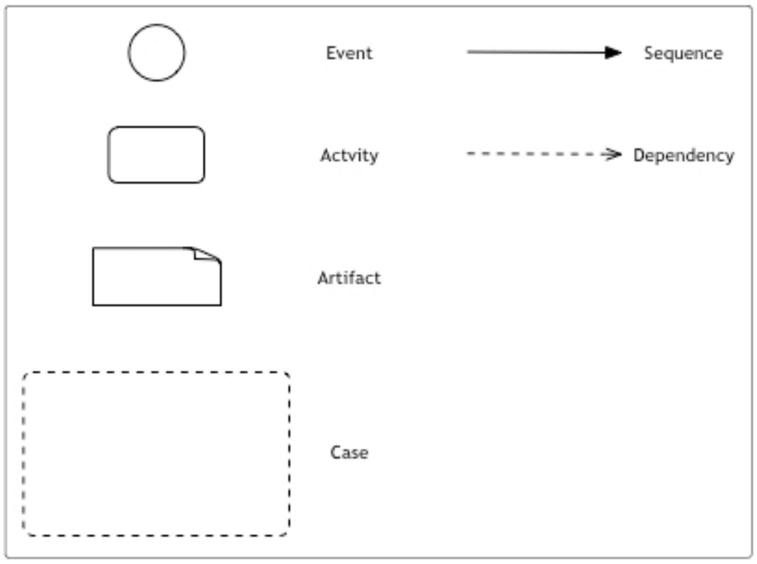

Figure 2. Modeling notation

of type *Dependency* or *Sequence*. While the former links a node with an artifact, the latter enables the connection between two nodes. A sequence definition is further classified in *Mandatory Sequence*, *Optional Sequence* and *Simple Sequence*.

The result of the modeling phase consists in an *annotated case model diagram*: a case model enriched by assertions derived from a specific ontology.

2.2.3 Operational Semantics

Given an annotated case model diagram, it is then possible to define its operational semantics in terms of an annotated case system.
An event is denoted by a state and, thus, its annotations become the annotations of the related state. The same translation applies for activities. Each case behaves as an automaton controller for its elements. In particular, transitions among internal elements adopt guards that are satisfied when the case has reached its intermediate state. The annotations are available in the intermediate state that composes the automaton. A mandatory sequence links two different states, where the final state can assume the initial state of an event, an activity or a case, and where the start state is an event, an activity or a case. Annotations become a guard for that transition. Simply speaking, the guard, if satisfied, enables the transition to the state at the other part of the relationship. The same applies for optional sequences. For a mandatory sequence the transition between the two states must be performed. For an optional sequence the transition could not be completed. A simple sequence is denoted by a transition between two states, where the first state can assume the final state of an event, an activity or a case. It cannot be annotated. When an artifact has no relationships, its annotations become the annotations of each state of the overall system. When an artifact is an input for a node, it means its annotations become available as guards on the transition that starts on the initial state of that element. When an artifact is an output for a specific element, annotations are included in the final state that belongs to the associated element.

2.2.4 *Support Functionalities*

On top of formal methods we provide two functionalities: *verification* and *selection*. Such functionalities are available both at design-time and run-time and support users during their tasks.
At design-time, using verification techniques, it is possible to discover problems before the design is enacted. In particular, we are able to check the correctness of models against dead-activities, conflicts, etc. Selection, instead, enables the reuse and auto-completion of process fragments included in ad hoc repositories.
Concerning run-time phase, the verification mechanism enables to monitor an instance during its life-cycle. In particular, for each specific instance, it is possible to track its status and verify if it has reached an inconsistent one. Furthermore, it allows to enforce correct changes on the executing instance. Concerning selection, it consists in recommending services that can be used to fulfill a specific result. This in order to improve intra- and inter-collaboration.

3 Analysis and discussion of main results

The main results presented are original in various aspects. First of all, we defined a formal model based on parametric state transition systems enriched by semantic annotations. Together with the introduction of annotated case systems, we also provided an algorithm based on Model Checking techniques. Both are completely new and extend previous results presented by Emerson and Kahlon in [13] and Di Pietro et al. in [14]. In addition, the proposed algorithm was applied to verification and selection problems both at design-time and run-time. This in order to provide functionalities to support users during their tasks. Finally, it is interesting to note that our approach turns out to be one of the few works, together with the two proposed by van der Aalst et al. (see [7] and [8]), that provides a systematization of the aspects related to BPM and CM.

The obtained results were applied to a Healthcare scenario. In particular, taking into account how radiological visits are executed within Hospitals, we validated different aspects introduced in our work.

4 Conclusions

This work investigated the possibility to integrate BPM and CM features with semantic aspects. The idea of merging both approaches is not unknown in the business management scenario. Thus, it is possible to represent and execute processes where users require a certain degree of freedom and can apply their expertise. Whereas it is needed, constraints are applied to enforce specific execution flows. Furthermore, semantic annotations accomplish to different goals. Assertions are used to express pre- or post-conditions on elements, retrieve process fragments, improve interoperability, etc.

A theoretical systematization for describing annotated case systems has been provided. Model Checking enables the verification of case systems against correctness properties expressed through indexed and annotated temporal specifications.

On top of the formal discussion, this work offers different tools for describing and executing case processes. In particular, introducing a modeling language and support functionalities, we are able to provide better guidance to users.

References

[1] S. Kemsley. *The changing nature of work: From structured to unstructured, from controlled to social.* in BPM, 2, 2011.

[2] C. Di Ciccio, A. Marrella and A. Russo. *Knowledge-intensive Processes: An Overview of Contemporary Approaches.* Knowledge-intensive Business Processes, 33, 2012.

[3] W. M. P. van der Aalst. *Challenges in business process management: Verification of business processing using petri nets.* Bulletin of the EATCS, vol. 80, pp. 174-199, 2003.

[4] J. Koehler, G. Tirenni and S. Kumaran. *From business process model to consistent implementation: A case for formal verification methods.* In EDOC, pp. 96-, 2002.

[5] J. Zhang and H. Wang. *A pi-calculus-based business process formal design method.* Lectures Notes in Computer Science, pp. 347-356, 2007.

[6] J. C. van Grondelle and M. Gülpers. *Specifying flexible business processes using pre and post conditions.* In PoEM, pp. 38-51, 2011.

[7] W. M. P. van der Aalst, M. Weske and D. Grünbauer. *Case handling: a new paradigm for business process support.* Data Knowl. Eng., vol. 53, no. 2, pp. 129-162, 2005.

[8] W. M. P. van der Aalst, M. Pesic and H. Schonenberg. *Declarative workflows: Balancing between flexibility and support.* Computer Science - R&D, vol. 23, no. 2, pp. 99-113, 2009.

[9] K. Kaan and Dr. W. Van Erde. *Modeling support for the case management paradigm and expectations management in process innovation.* 2005.

[10] P. F. Drucker. *Landmarks of Tomorrow: A Report on the New Post-Modern World.* New York: Harper Colophon Books, 1959.

[11] OMG. *Business process model and notation.* http://www.omg.org/spec/BPMN/2.0/2011, 2011.

[12] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi and P.F. Patel-Scheneider. *The Description Logic Handbook: Theory Implementation, and Applications.* Cambridge University Press, 2003.

[13] E. A. Emerson and V. Kahlon. *Reducing model checking of the many to the few.* in In 17th International Conference on Automated Deduction (CADE-17), pp. 236-255, 2000.

[14] I. Di Pietro, F. Pagliarecci and L. Spalazzi. *Model Checking Semantically Annotated Services.* Software Engineering, IEEE Transactions on, vol. 38, no. 3, pp. 592-608, 2012.