ESProNa: Constraint-Based Declarative Business Process Modeling

Michael Igler*, Paulo Moura†, Michael Zeising‡, Stefan Jablonski§

*University of Bayreuth
Chair for Applied Computer Science IV
Bayreuth, Germany
michael.igler@uni-bayreuth.de

†Dep. of Computer Science
University of Beira Interior, Portugal
Center for Research in Advanced Computing Systems
INESC Porto, Portugal
pmoura@di.ubi.pt

‡University of Bayreuth
Chair for Applied Computer Science IV
Bayreuth, Germany
michael.zeising@stmail.uni-bayreuth.de

§University of Bayreuth
Chair for Applied Computer Science IV
Bayreuth, Germany
stefan.jablonski@uni-bayreuth.de

Abstract—In this paper we describe how declarative process modeling together with ontologies can be used to build complex clinical process models. Our approach supports the definition of functional, behavioral, organizational, data and operational process perspectives, resulting in an expressive and flexible modeling language. We use constraints for representing interprocess dependencies and constraint propagation for finding which processes are executable in user selected or given scenarios. Knowledge about the organizational perspective of a clinical ontology can be represented and imported from RDF files for interfacing with other applications. We implemented our approach in ESProNa, a Logtalk application running on SWI-Prolog extended with the CLP(FD) constraint library and the N3 parser Henry.

I. INTRODUCTION

Global markets have raised the economic pressure on companies noticeably in past years. Products, services, and software from countries with lower wages have become a fierce competitor for business organizations. As a consequence, efficient development of products and services become a critical factor for their success [1]. A basic prerequisite in improving business productivity is a detailed understanding of the processes and value chains within an organization. Business process modeling aims to both document the flow of activities within a company and identify opportunities for process optimization. Processes quality is another key factor in successful organizations: [2] states that product and software quality are proportional to process quality in business organizations. Two major standards have been established to assess the quality and maturity of processes: the CMMI [3] model, developed by the Software Engineering Institute (SEI), and the ISO/IEC 15504 standard [4] (a.k.a. the SPiCE standard).

Some of the key features for effective process modeling software are: flexibility, expressiveness, and separation of concerns. These features are considered some of the most challenging tasks in current process modeling research. Flexibility is an increasingly requested feature for process modeling languages, making them more appropriate to model the complexities of actual applications [5], [6]. It allows the process modeler to minimize the strictness of process models and to more easily integrate changes in requirements into existing process models. It also gives the process executor the freedom to choose between different processes at execution time. Expressiveness and separation of concerns dictate how a process modeler can describe all perspectives that make up a process [7], [8]. These features aim to make it possible to bring together and represent all contributors (including persons, tools, and data) that collaborate in a business process, allowing fine-grained representation of each contributor while keeping their orthogonality.

In this paper, we present a declarative approach to process
modeling and execution, implemented in the ESPProNa application. Using a declarative approach lets the modeler focus on representing the process constraints and to delegate process execution simulation (including the search for viable and preferred process execution scenarios) to the ESPProNa constraint-solving engine. This approach simplifies process modeling and improves the readability of process models by third-parties by avoiding imperative control flow models that emphasize how instead of what. In ESPProNa, flexibility is supported by allowing the representation of non-deterministic process models. Both expressiveness and separation of concerns are supported by allowing the representation of relevant process information using explicit and orthogonal process perspectives, following the Perspective Oriented Process Modeling (POPM) approach [9].

In this section we presented key concepts that guide the development, evaluation, and comparison of process modeling software. The rest of the paper is organized as follows. In Section II we describe how the separation of concerns can be accomplished in process modeling by using a disjunctive categorization of a process into several perspectives. In Section III we give an example of a clinical process model that illustrates the complex demands of process modelers to a modeling language. This example is used in the following sections. In Section IV we provide a detailed description of the ESPProNa framework, which implements the research results presented in this paper. In Section V we compare ESPProNa with related process modeling software.

II. PROCESS PERSPECTIVES

A process can be described from different orthogonal perspectives, following the POPM approach:

- The functional perspective describes the process itself. In particular, the number of times that a process must or can be executed. Complex processes can be decomposed into a set of steps or sub-processes.
- The behavioral perspective describes the order in which the process steps have to be executed.
- The organizational perspective defines persons that are responsible for the execution of a given step.
- The data perspective describes the data consumed and produced by a process step. Taken together, the data input and output of each process defines the data flow of a process model.
- The operational perspective defines tools and systems that support the execution of a process step respectively generating data.

By combining these perspectives, a comprehensive description of a process can be accomplished.

III. A PROCESS MODELING EXAMPLE

As a motivating example, we present a process model on clinical surgery, extracted from an industrial project on recording clinical procedures. We also use this example to introduce the graphical notation used in our declarative approach to process modeling.

Figure 1 presents the symbols used for representing each perspective on the left side. The right side of the figure details some of the informal constraints associated with some of the perspectives. For example, in the detailed information regarding the organizational perspective, we state that an assistant doctor of the X-Ray group is required for this process. Processes are represented by rectangles with round corners. In the upper half, there is a short process description, with a Process Identifier (PID) on the lower left side (a unique integer). The PIDs provide a simple solution for referencing a process from other processes. On the lower half of the process, we list all the perspective constraints. It is possible to list more than one constraint for every perspective or to omit a perspective completely.

Constraints can be marked with the tags S (to start), F (to finish), or A (to abort). These tags are used to convey additional information on constraints (for any perspective) on their influence to start, finish, or abort a process. This additional information helps to make process modeling more agile and amenable to actual business scenarios. For example, we can use these tags to represent that the order in which certain process are executed can depend on the state of other processes.

Figure 2 uses our graphical notation to illustrate how the X-Ray and Cardiology teams may collaborate in performing an Ablation of Atrial Fibrillation. We use the five processes pictured on this figure to explain in detail how the different perspectives described in Section II can be used to model this clinical procedure.

A. FUNCTIONAL PERSPECTIVE

The functional perspective is identified by the clipboard symbol. Different notations are available to express how often a process must or can be executed. The min..max notation (see e.g. process PID 1) can be used to express the minimum and maximum number of times that a process can be executed. It is also possible to express that a process must be performed...
e.g. as often as another process. These kinds of constraints are particularly useful in producer/consumer scenarios. In our example, the Visualization process is constrained to be executed the same number of times as the Record 3D Image process (i.e. a visualization is made for every recorded image).

### B. Behavioral Perspective

The behavioral perspective is identified with a small graph symbol. In our example, the process Record X-Ray Sequence includes a behavioral constraint stating that another process, PID 1, must be marked as done. The 2S tag means that this constraint only influences the ability to start the Record X-Ray Sequence process. This tag is also used e.g. in the behavioral constraint of process PID 5. Here we declare that process PID 4 must have been started so this process can be started.

### C. Organizational Perspective

The organizational perspective is represented by a person shape. Consider the process Prepare Procedure in our example. The first organizational constraint states that this process must be started by an MTA (Medical and Technical Assistant) of the X-Ray or Cardiology group. The second constraint states that the agent who can finish this process (2F) must be the same agent who started it. In the organizational perspective, we use agent as a generic term for describing people, machines, or computers. This concept has been extensively discussed in [10]. The Prepare Procedure process illustrates a combination of flexibility in starting the process (the choice of persons of different groups) with a constraint on the person who can finish it. Another important feature of our work is that is possible to dynamically assign agents to a process that depend on the state of other processes. We can see an example of this feature in the second organizational constraint of the processes PID 2 and PID 3. The constraints state that each process must be started by the same agent who did or will do the other process. This feature allows a flexible binding of agents to a process that relates to other processes which have already started or are already done.

### D. Clinical Ontology

Organizational information, such as describing which persons are part of a certain group or have a certain role, are stored within our framework using RDF data in the N3 notation [11]. An N3 document encodes a set of statements whose meaning is the conjunction of the meaning of all statements. Statements are encoded using the format x p y, which asserts that the relation p holds between x and y. N3 is essentially equivalent to RDF [12] in its XML syntax, but easier to write down.

Figure 4 shows a graphical excerpt of an exemplary clinical organization: Charles, Peter, John and Claire are staff members of the X-Ray department playing the roles of, respectively, Medical Superintendent, Assistant Medical Director, Assistant Doctor, and Medical Technical Assistant. Similar organizational structures can be found in the Cardiology department: Medical Superintendent (Jacob), Assistant Medical Director (Hugo), Assistant Doctor (Jack) and Medical Technical Assistant (Kate). The associations between each person and its role(s) are expressed by the org:plays relation. The foaf:member relations express the associations between persons and departments.

Information about which person supervises another person is encoded within the role that a person holds. Figure 3 shows how this hierarchical structure is encoded in an N3 statement. This hierarchy can be found in the middle of Figure 4. The MedicalSuperintendent node is connected to the AssistantMedicalDirector node through a labeled org:supervises arc which means that the Medical Superintendent supervises his employed Assistant Medical Director. Analogue relations between the remaining employees can be found in the data graph. The relation org:supervises is transitive. Therefore, the Medical Superintendent also supervises the Assistant Doctor and the Medical and Technical Assistant. The role of every person and the supervising relation between roles allows us to infer a predicate supervisorOf between persons.

### IV. IMPLEMENTATION

ESProNa is implemented as a Logtalk [13] application, using SWI-Prolog as its back-end compiler, extended with the CLP(FD) constraint library [14] and the N3 parser Henry [15]. Logtalk object-oriented features are used to represent processes, constraints, agents, and other entities found in
process modeling. The CLP(FD) library is used for evaluating functional process constraints. As described in the previous section, the N3 parser Henry is used for building and querying organizational information. In the following sections we present a detailed description of the ESProNa framework.

A. Process Representation

Processes are represented using Logtalk parametric objects [16]. The identifiers of parametric objects are compound terms containing logical variables, which act as parameters. In our case, the parameter of each process object is used as the process constraint variable. The use of these parametric object identifiers when representing lists of process dependencies allows us to restore shared constraint variables at runtime in order to model sets of inter-dependent processes. As an example, consider the object representation of the process PID 4 in Listing 1. The functional dependency of this process, expressed by the predicate functional_constraint/2, includes a constraint that will connect at runtime the PID 4 and PID 3 constraint variables.

Perspective constraints are represented using Logtalk categories [16], which are fine-grained units of code reuse. Categories allow us to retain at the implementation level the orthogonality of the different perspectives. All categories representing perspective constraints implement a common protocol, which simplifies the implementation of the process methods. An example of the organizational perspective constraint category can be found on Listing 2.

B. Process Actions and Constraints

During the execution of a process model, different actions can be performed on a process. For example, a process can be started, finished, or aborted. ESProNa, however, is not restricted to a fixed set of process actions. The process modeler can define custom actions. An example could be defining an action allowing a process to be suspended.

Before an action can be performed, every existing process constraint must be checked. For example, users often need to know which processes are executable in a selected scenario. Given a list of processes, the list of possible actions can be obtained using constraint propagation over the corresponding set of process constraints. The list of valid actions that satisfy the constraints can then be presented to the user.

C. ESProNa Architecture

The different POPM perspectives are implemented using categories [17], which are imported into the object process. This object is used as a prototype for the objects that implement the actual business processes.

Every POPM perspective implements the constraint protocol, which specifies a conform/4 predicate (an implementation example can be found on Listing 2). This predicate is used for checking process constraints in a given scenario.

The process object implements two protocols that specify a set of predicates for example, startable/3 that are used for checking if an action is possible in a given scenario. These predicates call the conform/4 predicate, in order to check the existing constraints.

In order to be able to execute process models, it is necessary to maintain the state of all processes at each execution step. In the next sections, we provide a detailed description how process models states are represented and how this state representation is used by the planning component of the ESProNa framework.

D. Representation of Execution States

In order to ensure clean and logical semantics and preserve the possibility for backtracking over previous execution states, the description of the individual state of all processes is stored in state objects. There is an entry in this state object for each process. This entry is represented by a compound term and stores the PID, the process history, and a boolean flag that signals if the process has been completed. The process history is represented as a list of entries of the form ProcessInstanceID-Status-Agents. ProcessInstanceID is an integer that is automatically incremented, representing the instance of a certain process. If e.g. a process can be performed two times, then two possible instances of the process can be generated.
ESProNa code for process PID 4

```
:- object(pid_4(_),
    extends(process)).

    functional_constraint([pid_3(PID_3)], (PID_4 #= PID_3)) :-
      parameter(1, PID_4).

    behavioral_constraint(start, State, _, [pid_1(_)],
      (pid_2(_)::done(State))
    ).

    organizational_constraint(start, _, _, [Agent],
      (group('XRayDepartment')::member(Agent),
       person(Agent)::plays('AssistantDoctor'))
    ).

    organizational_constraint(finish, State, Instance, [Supervisor],
      (pid_4(_)::instance_agent(State, s, Instance, StartAgents),
       list::member(StartAgent, StartAgents),
       person(Supervisor)::supervisorOf(StartAgent))
    ).

:- end_object.
```

Listing 1

```
:- category(organizational_constraint,
    implements(constraint_protocol)).

:- public(organizational_constraint/5).

    conform(Action, State, Instance, Agents) :-
      bagof(
        Constraints,
        ::organizational_constraint(Action, State, Instance, Agents, Constraints),
        List
      ),
      meta::map(call, List)
    ; 
      
    :- end_category.
```

Listing 2

Organizational constraints category

**Status** represents the current status of the process using a status code. **Agents** is a list of agents that performed the action (an action process can be performed by several agents; this is called the n-eye principle [10]). For example, when the process PID 1 is started by Claire of the X-Ray department, the entry 1-s-[Claire] is added to the history list (assuming this was the first action applied to the first instance of the process).

Entries in the history list can be used during the evaluation of constraints. For example, we may want to express that, when a person aborts a process, only the person’s supervisor can restart it. This requires access to the process history list in order to retrieve information about the agent that aborted the process. This information can be used to query the organizational information and retrieve the persons who can restart the process.

**E. Planning**

ESProNa features a planning component, specifically developed for the domain of process modeling and execution, which serves as a navigation system for the process executor. This planning component is based on the concept of state-space search and makes use of the state objects described in the previous section. We use modified versions of the state-space algorithms provided with the Logtalk distribution in order to allow the search of paths between two arbitrary process execution states. This modification allows the process executor to start from a process that is not in the list of directly executable processes. The planning component allows access to all possible states that a process executor can reach by performing different actions. As an example, consider the sequence of states in Figure 5. The process executor is interested in the state where the process Edit 3D-Image/Visualization (PID 4) becomes executable. That is, the sequence of steps that have to be performed in order to reach a state where
the process can be started. Furthermore, the process executor wishes to know which agents can start the process.

From the first state to the second state, a labeled arrow is drawn which represents the state transition. It depicts the first action (s) that is performed on the process pid_1 by Claire (she is an MTA and member of the X-Ray department, as constrained in the process model). In the next state transition, the process pid_1 is finished (f). The only agent who can perform this action is Claire of the X-Ray department as the constraint 2F: Agent = self.startBy(Agent) in the process model restricts this action to her. In the resulting state (3rd from top) we can see in the history list of process pid_1 that an additional entry has been added. This entry represents the action (action 1 finished (f) by Claire) that has been performed. The completed flag of process PID 1 is set to 1. As modeled in process PID 2 and PID 3, this fact is a mandatory constraint for the evaluation of the behavioral constraint in PID 2 and PID 3: only in this case the mentioned processes can be executed (started).

V. RELATED WORK

The current approaches to process modeling and planning can be categorized as either imperative approaches or declarative approaches. In this section, we compare our work with some representative implementations of both imperative and declarative approaches. [18] gives an overview on declarative vs. imperative process modeling approaches. It was our design goal to support beside the benefits of declarative approaches the opportunities of imperative approaches which are: the life of each process can be described by its state and the possibility to compute and navigate between arbitrary process states.

A. Imperative approaches

Imperative approaches to process modeling have their roots in imperative programming languages, adopting concepts such as conditionals (if-then-else) and loops (for, while, repeat until) in order to represent the execution flow in process models.

Current imperative approaches to business process modeling suffer from two known limitations. First, lack of flexibility of the process execution scenario. Controlled deviations are not supported natively and must be modeled explicitly. This leads to complex process models that are hard to understand and to communicate. Second, the lack of separation of concerns and limitations when representing different perspectives of the process model hinders comprehensive representations of complex business processes.

Figure 6 illustrates the first problem in imperative-based process modeling using an example found on [19]. Three process, A, B, and C, must be executed in that exact order. Process B can be executed between one and three times, depending on a user decision. This simple requirement results in a complex imperative model where each possibility is unfolded and modeled explicitly. Changes in requirements, for example, allowing process B to be executed from eight to ten times, can quickly render the process modeling impractical. This low-level, control-flow based approach hinders readability. The process modeler knows the business meaning of the model from building it; others may have to infer this meaning from the model itself. This inference is harder the more complex the model.

[20] makes an evaluation of the suitability of the imperative modeling language BPMN [21]. It evaluates the modeling languages against the workflow patterns from [19] concluding that there exist inherent difficulties in applying a language that does not have commonly agreed-upon formal semantics nor an execution environment. Although there is a mapping from BPMN to the execution environment BPEL [22], closer inspections show that this mapping is only partial, leaving aside models with unstructured topologies.
The research described in [23] does a comparison of business modeling and execution languages coming from the open source area. It concludes that open source systems like jBPM [24] and OpenWFE (now ruote 2.1) [25] are geared more towards modelers who are familiar in programming languages than towards domain experts.

YAWL (Yet Another Workflow Language) [26] is a workflow language that makes use of so called high-level Petri nets to refer to Petri nets extended with color, time and hierarchy. The definition of YAWL presented in [26] only supports the control-flow (behavioral) perspective. Therefore newYAWL [27] has been developed to provide support for the control-flow, data and resource perspectives. Nevertheless the functional and organizational perspective are still neglected.

B. Declarative approaches

DECLARE [28] is a constraint-based system, developed at the University of Eindhoven, that is focused on modeling constraints between processes. It supports the behavioral and the functional perspectives of POPM. DECLARE uses the ConDec [29] modeling language. Modeled constraints in ConDec are translated to a Linear Temporal Logic (LTL) formula. There is an automaton generated for every specific constraint in order to verify it. Furthermore, an automaton is also generated over all constraints. The support for the organizational perspective in DECLARE is, however, limited as hierarchical structures cannot be modeled. A planning component that can be consulted for advice during execution phase is also absent.

EM-BrA²CE (Enterprise Modeling using Business Rules, Agents, Activities, Concepts and Events) is a framework for unifying vocabulary and execution models for declarative process modeling [30]. The vocabulary is described in terms of the Semantics for Business Vocabulary and Rules (SBVR) standard and the execution model is presented as a Colored Petri Net (CP-Net). EM-BrA²CE also follows the same concept as we use in this paper to specify a state space transition relation based on rules. However, functional and operational perspectives are not supported in this framework. Furthermore, the process modeler has no possibility to graphically “model” business processes. Instead, every process must be described in the form of the mentioned Business Vocabulary. This slows down re-reading of process models by different users or the process modeler itself after some period of time.

Etalis [31] is a logic-based language for Complex Event Processing (CEP). CEP works by processing multiple events with the goal of identifying meaningful event patterns. Detected event patterns and situations are then used to trigger response actions which in turn may trigger other events. In contrast with ESPreNa, Etalis is a reactive system that is event or data-driven. When certain events occur, different actions are automatically performed, which can trigger other events. In ESPreNa, we choose to shift control during execution to the user. The constraints are specified by the process modeler and ESPreNa makes sure that these rules are abided by the user.

The Freeflow [32] prototype shares with ESPreNa the use of constraints for building declarative process models. Freeflow constraints represent dependencies between states of different activities. An activity for example can enter a specific state only if another activity is in a certain state (e.g. inactive, active, disabled, or enabled). The architecture of Freeflow makes a separation into three interacting parts: the process manager, inter-actors, and events. The process manager acts like a database system, managing the records of current process instances and performing process-specific manipulations. An inter-actor is a component of the system which deals with interaction and process representation issues. The link between these components is a generalized event mechanism. This mechanism is used for process coordination in alternative to the direct definition of communication patterns between system components. We could not infer from [32] which of the POPM perspectives are implemented or how a process is modeled. In particular, is not clear how the key process modeling features described in Section I are fulfilled.

VI. Conclusions

The main contribution of this paper is a declarative process modeling approach that uses a combination of object-oriented representation of processes and constraint propagation to support the key features for effective process modeling: flexibility, expressiveness, and separation of concerns.

The use of the object-oriented logic programming features of Logtalk allows processes and their constraints to be expressed declaratively in an easily extendable framework, which reflects the orthogonality of the POPM perspectives at the implementation level. This separation of concerns, combined with constraints and constraint propagation, translates to an expressive process modeling environment. In particular, the use of constraints makes it possible to only constrain process execution as strictly necessary. This results in more freedom when executing process models but also increases the risk of overloading the user with a large number of possible execution scenarios, most of them of limited value for the actual business modeling. We solve this problem by using...
a planning component that helps the process executor in reaching user-defined or given scenarios.

Finally, separation of concerns is achieved by explicit and orthogonal representation of all five process perspectives found on POPM [9]. In particular, the use of RDF data graphs for describing individuals and relations in business organizations helps to integrate our framework with business information systems and to avoid redundant representations of staff organization data.

Acknowledgements

We thank Jan Wielemaker, Markus Triska, and Yves Raimond for their groundwork on, respectively, SWI-Prolog, the CLP(FD) library, and the N3 parser. This work is partially supported by the FCT research project MOGGY – PDT/C/EIA/70830/2006.

REFERENCES


