Improving Collaborative Business Process Execution by Traceability and Expressiveness

Michael Zeising
University of Bayreuth
Bayreuth, Germany
michael.zeising@uni-bayreuth.de

Stefan Schönig
University of Bayreuth
Bayreuth, Germany
stefan.schoenig@uni-bayreuth.de

Stefan Jablonski
University of Bayreuth
Bayreuth, Germany
stefan.jablonski@uni-bayreuth.de

***Abstract*—The declarative modeling approach promises to be a suitable means for the description of rather unforeseen and less rigid business processes. However, today's approaches for the execution of such declarative processes lack certain essential capabilities. One of those is *traceability* which means that the actions proposed by the execution engine are explained and justified. Furthermore, *expressivity* is mostly limited to static temporal dependencies that lead to a simple temporal ordering of process steps without a proper connection to process perspectives like incorporated data, agents performing the work and utilized tools. Finally, due to their core principles, today's declarative execution engines suffer from *scalability* issues. This article outlines a concept and a prototypical implementation of an execution engine that aims to solve the above issues and integrates with current business process frameworks.**

Keywords-business process management; process execution; declarative; constraints; traceability; expressiveness; scalability

I. INTRODUCTION

Current research in the field of business process management has pointed out that a useful process-aware information system must exhibit a trade-off between supporting people in achieving their goals and granting them as much freedom as possible at the same time [1]. In this context, a business process that restricts people to a lesser extent is characterized as a "flexible" process [2]. Concerning process modeling, a flexible process must comprise more possible paths than a rigid one.

Following the terminology of programming languages, there are two paradigms of describing business process models: the *imperative* and the *declarative* style. The imperative way corresponds to imperative or procedural programming where every possible path must be foreseen at design time and encoded explicitly. If a path is missing then it is considered not allowed. In declarative modeling, on the other hand, only the undesired paths and constellations are excluded so that all remaining paths are potentially allowed and do not have to be foreseen individually. As the so-called flexible type of business processes incorporates many often unforeseen paths, the declarative approach is best suited for it [3].

If business process execution involves human participants then the supporting system should be treated as a decision support system instead of a means of automation. As a consequence, an according system should propose actions and support them but it should never enforce them [4]. This requirement goes along with the call for process management systems "to provide directions and guidance rather than enforcing a particular route" whereas they are compared to navigation systems [5]. An important characteristic of decision support is explanation and so-called meta-knowledge [6]. Proposals made by the system need to be justified and explained so that sound choices can be made. For process execution, this means that certain proposed actions must be marked as recommended and that discouraged actions can be traced back to and explained by the according parts of the business process model. A declarative and therefore less rigid form of business processes leads to a greater amount of possible paths. So, especially in the declarative area, the capability of explanation comes into value.

Strongly connected with traceability is the expressivity of process modeling languages. The process execution engine directly affects and limits the expressivity of the languages it interprets. Declarative modeling is based on constraints that relate events of the process and exclude or discourage from certain correlations. Both constraints and events must be able to involve all the perspectives of a business process like, e.g., incorporated data, agents performing the work and utilized tools [7]. On this way it becomes possible to express realistic correlations like, e.g., the actual performing agent of a step affecting the type of data used in another step [8]. Another important means of enhancing expressivity is the support for different modalities. A modeling language greatly benefits from a distinction between optional and mandatory statements.

With the great flexibility of a declarative description comes the issue of complexity. The principle implies that virtually every possible path must be evaluated during execution. Depending on the size of the model and the number of involved perspectives, this results in a huge amount of possibilities and the so-called state explosion. In order to maintain scalability, a declarative execution engine must avoid this limitation.

In this article, an execution engine for declarative business processes that addresses the above issues is described. Instead of interrelating steps by unconditional temporal constraints, the engine allows for constraints to depend on multiple perspectives of a business process like, e.g., data, agents and tools. Together with the support for qualifying constraints by different modalities ("must" and "should"), the proposed engine allows for more expressive process modeling languages. Due to the support for different modalities, the engine is able to recommend certain actions or to discourage from them. Whenever an action is advised against, the engine may explain and justify this decision by tracing it back to the process constraints, i.e., modeling constructs that would be violated. Therefore, the engine offers traceable business process execution. Process models are interpreted in a stepwise manner instead of precalculating all possible paths prior to the execution. On this way, the engine scales better than complete approaches and is capable of executing business processes of real-world size and complexity.

This work is kindly funded by "Europäischer Fonds für regionale Entwicklung (EFRE)"

COLLABORATECOM 2012, October 14-17, Pittsburgh, United States

DOI 10.4108/icst.collaboratecom.2012.250345

Section 2 outlines the related work on declarative business process execution and highlights the approaches' drawbacks as well as adopted aspects. Section 3 summarizes the contribution of this article and delimits its scope. Section 4 guides through the principle ideas of the concept behind the execution engine. Section 5 shows how the concept is implemented on top of a business logic framework. Section 6 presents a running example of a business process modeled for and executed by the engine. Finally, section 7 concludes the article, goes into the current limitations of the approach and provides an outlook on planned future work.

II. Related Work

The most recent approach in the field of declarative process management is the *Declare* framework [9]. It is based on linear temporal logic (LTL) and therefore allows for relating process steps by temporal and existential constraints. Pattern (1), e.g., claims that if process step A is performed then step B must be performed eventually.

$$\Box(A \rightarrow \Diamond B) \tag{1}$$

These constraints may not contain statements on data, agents or tools. The only way of relating the temporal order of steps to these perspectives is to make the constraints depend on certain conditions. Such a conditional constraint only applies if its condition evaluates to true. Though a condition could then contain statements on data, agents and tools, the actual constraint remains limited to temporal order and existence of steps. The other perspectives cannot be constrained, which reduces the expressivity of the supported process modeling languages.

For execution, the LTL formulae of a process are transformed into a finite state automaton which will then accept every trace of events that complies with the formulae. In order to reach a technically feasible size of the automaton, only the completion of a step is considered.

Though a distinction between optional and mandatory constraints is made in the theoretical preliminaries, distinct modalities are not supported because only one automaton is generated for the mandatory formulae. Both the LTL formulae and the automaton must be transformed and reduced for necessary optimization reasons. Due to that, it becomes impossible to draw a connection between the automaton's transitions and the originally modeled constraints. Therefore, Declare cannot support traceability during execution as the proposed actions cannot be explained. In spite of the simplifications and reductions, Declare suffers from scalability issues. Process models of realistic size lead to large automata which have to be generated completely before execution.

There are several approaches that are very similar to Declare. In the work of Sadiq *et al.* [10] and also in the work of Wainer *et al.* [11], temporal constraints like, e.g., *serial*, *order* and *fork* are used to relate steps. As for Declare, these constraints may neither depend on nor influence perspectives like data, agents or tools and modalities are not supported either.

A different idea was pursued with *EM-BrA²CE* where business processes were modeled and executed on the basis of business rules [12]. The use of a first order rule language promises high expressivity and is also pursued in the work at hand. However, it is unclear to what extent this approach was realized and many details are not described. As a result, it cannot be assessed to what extent traceability is supported, how expressive the process modeling language effectively is and how the approach scales.

The *ESProNa* engine realized the execution of declarative business processes covering multiple perspectives and different modalities [13] [14] and therefore already supported highly expressive process modeling languages. However, this monolithic prototype did not support traceability. ESProNa can be viewed as a predecessor of the approach at hand.

The *Process Virtual Machine* (PVM) defines a common model for the execution of imperative graph-based process modeling languages like BPMN and EPC [15]. It features many crucial concepts like, e.g., composite processes, asynchronous execution, transactions, persistence and wait states. The PVM model forms the basis of the *JBoss jBPM* engine. It mimics a token-based interpretation by moving execution pointers along the process graph which limits its scope to the execution of imperative processes. The objective of the approach at hand is to adapt the PVM's concepts for the execution of declarative processes in a reasonable way so that the two architectures may be integrated in the future.

III. Summary of the Contribution

In this work, a concept and a prototypical implementation of a constraint-based process execution engine is outlined. This engine is able to execute declaratively modeled business processes and addresses the issues of traceability, expressivity and scalability. Instead of only constraining the temporal order of process steps, it allows for constraints to influence multiple perspectives like, e.g., incorporated data, performing agents and utilized tools. Together with a distinction between optional and mandatory constraints, it enables the use of modeling languages that are more expressive and cover more aspects of a business process than today's approaches based on simple temporal constraints. Actions proposed by the engine are justified and explained by the original process model constraints. Therefore, the engine realizes traceable execution of business processes and allows for sound choices to be made by the process participants. Instead of pre-calculating all possible paths before execution, the engine only evaluates the respective next action. Thus, scalability is maintained and the system is suitable for business processes of realistic size.

The presented work is part of a broader project that aims at a modular multi-paradigm platform for business process execution named *AI4 Process Navigation*. The engine at hand is intended to be for declarative processes what the Process Virtual Machine (PVM) is for imperative ones [15]. The project's overall purpose is to better understand the nature of business process execution and its connection to process modeling. Fig. 1 illustrates the pursued architecture of this framing project. The *imperative execution core* as well as the *declarative execution core* described in this paper share a common infrastructural basis (*common foundation*). The support for different process modeling languages is built on top of the respective core.

The realization of these language modules is only touched in places by this contribution and is out of scope in principle. The aim of this paper is not to propose a new declarative process modeling language but a platform supporting the execution of a wide range of these languages.

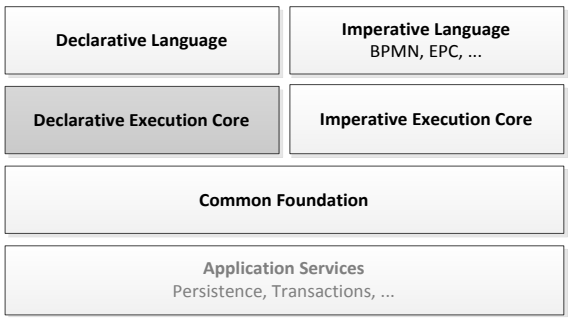

Figure 1. Pursued architecture of the framing project with the declarative execution core described in this contribution

IV. Concept of the Execution Engine

The core principle of modeling and interpreting a declarative business process is to constrain transitions in a state space. Each transition is connected to an event in time describing multiple perspectives at once like, e.g., "Bob finishes reviewing the CNC model using Autodesk® Inventor®". The current state of the process case is then determined by the trace (or "log") of past events. A process constraint usually refers to that trace like, e.g., "the sketch must be created before it can be reviewed" but also "the sketch should be reviewed by the supervisor of the person who created it". A declarative process model thus consists of the static entities like process steps, data items and agents and a set of constraints referring to them. Feasible next transitions or rather events are derived from evaluating the process constraints on the basis of the current event trace.

A. *Process Constraint Solution*

The interpretation of process constraints is triggered whenever the state of the case changes. The first step of this procedure is to generate virtually all possible next activation events of the process case. For a model with the processes P, the agents A and the data items D, this results in $e = |P \times A \times D|$ events that could be potentially generated. However, depending on the solution strategy, only a sub-set of these events has to be considered in order to find feasible ones.

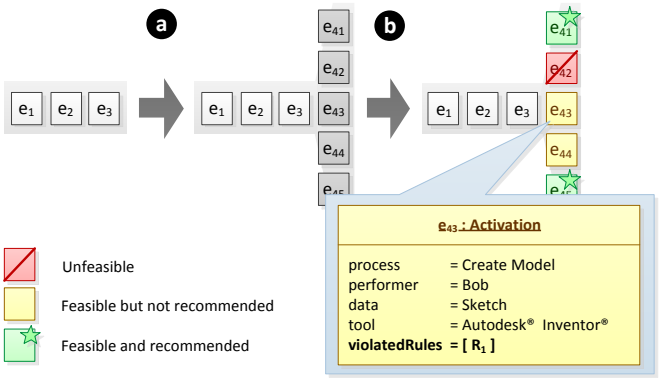

Figure 2. Stages of the evaluation: potential event traces are generated (a) and scored (b)

In the next step, each of the generated activation events is combined with the current trace of past events so that e possibly resulting traces are built. These possible traces are now *scored* on the basis of the process constraints. The result of scoring is that each potential next event is classified as *unfeasible*, *not recommended* or *recommended*. Unfeasible events are discarded and are considered no further. If an event is not recommended references to the constraints (rules) that it violates are attached to the event. Fig. 2 illustrates these stages of the evaluation procedure. The current trace of the process case consists of the events e_1 to e_3. Now the interpretation is triggered and the next feasible actions should be proposed. We assume that five next events e_{41} to e_{45} result from the according process model. One of these could be the event that "Bob" performs the step "Create Model" using "Autodesk® Inventor®" and consuming some "Sketch". The potential next events are now scored according to the process constraints. We assume that the events e_{41} and e_{45} do not violate any of these constraints so they are considered *recommended*. Event e_{42} violates a hard constraint so it is considered *unfeasible* which means that it must not occur. We assume that e_{43} violates a soft constraint R_1 so a reference to this constraint is attached to the event and it is considered *not recommended*. When proposing the according action to the process participant ("Bob" in this case) the devaluation of it can now be explained by tracing it back to the process model (constraint R_1 in this case).

A constraint is expressed in the form of a first-order logic expression x that refers to the event trace. The following expression, e.g., claims that if there is an activation event a_B for process "B" then there must be a completion event c_A for process "A" performed by "Alice". In other words this constraint ensures that process "A" has to be completed by "Alice" before process "B" can be activated:

$$\begin{gathered}\text{Activation}(a_B) \wedge \text{process}(a_B, \text{B}) \wedge \text{time}(a_B, t_B) \\ \rightarrow \\ \text{Completion}(c_A) \wedge \text{process}(c_A, \text{A}) \wedge \text{performer}(c_A, \text{Alice}) \wedge \\ \text{time}(c_A, t_A) \wedge \text{before}(t_A, t_B)\end{gathered} \tag{2}$$

The approach supports different modalities. As a result, constraints are divided into hard and soft constraints depending on whether they render a trace unfeasible or not recommended. So the next step is to assign a modality $m \in [\text{H}, \text{S}]$ to the constraint which results in a triplet (i, x, m) where i is a unique identifier for referring to the constraint, x is the original expression and m encodes the constraint's modality. The purpose of a *scoring rule* is to punish the violation of a constraint, i.e., the negation of the constraint and to devaluate the according solution. Assuming that the above constraint is a hard constraint, the scoring rule triplet for it is:

$$\begin{gathered}(\text{A\_By\_Alice\_Before\_B}, \neg x, \text{H}) \\ \equiv \\ (\text{A\_By\_Alice\_Before\_B}, \\ \text{Activation}(a_B) \wedge \text{process}(a_B, \text{B}) \wedge \text{time}(a_B, t_B) \wedge \\ \neg(\text{Completion}(c_A) \wedge \text{process}(c_A, \text{A}) \wedge \text{performer}(c_A, \text{Alice}) \\ \text{time}(c_A, t_A) \wedge \text{before}(t_A, t_B)), \text{H})\end{gathered} \tag{3}$$

The generated traces (and the respective events) can now be classified in the following way:

- A trace that violates one or more hard constraints is considered unfeasible and the according event will be discarded.
- A trace that violates no hard constraints but one or more soft constraints is considered feasible but not recommended. A reference to the violated soft constraints is maintained for traceability.
- A trace that violates no constraints is considered recommended.

As outlined above, the events of the feasible traces are now returned and lead to the activation of the according process steps. Every trace that is rendered not recommended because it violates soft constraints is equipped with the according rule reference. On this way, the actions proposed by the system become traceable because they can be explained by the original model elements. The engine may now make statements like "event e_1 should occur and event e_2 may occur but it should not because of r".

B. *From Events to a Process's Life Cycle*

The core execution engine is independent from the process modeling language. Therefore, it must not have any knowledge of different process step types like, e.g., human tasks, script tasks or service invocations. The engine only considers entering (activating) and leaving (completing) abstract process model nodes. These two transitions and the according events are called *activation* and *completion*. Whenever the state of the process case changes, i.e., when the case is initialized or a node was left (completed), the constraint solution part of the engine is asked for the next nodes to be entered, i.e., for the next feasible activation events. These new activation events are appended to the event trace and interpreted according to the type of process step to be activated. For a human task, e.g., activation means that the according work item should be scheduled in a work list. For a script task, it means that the script should be invoked. This interpretation of events is illustrated in Fig. 3.

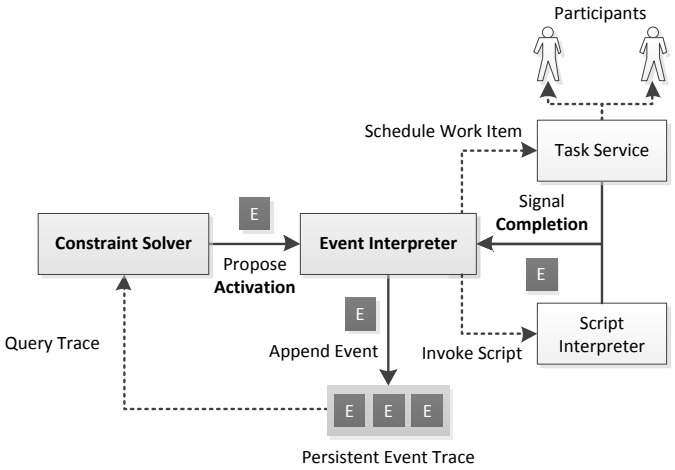

Figure 3. Activation events are proposed by the constraint solver while completion events are signalled by process step implementations

After activation, process steps traverse their respective life cycles which is reflected by different process behaviors. A human task, e.g., usually runs through various states like its delegation, assignment and so on. However, these sub-states are hidden from the core engine and are handled outside of it, e.g., by a human task service. Only the completion of a step is signaled back to the engine. The event is appended to the trace and the constraint solver is invoked again. On this way, the process model is interpreted incrementally which reduces the complexity compared to approaches that completely precalculate all possible traces. As a result, the approach scales better than complete approaches.

C. *Sub-Processes and Execution Pointers*

Regardless of whether a process model is being designed imperatively or declaratively, each process step may be composed of further steps. In contrast to the concepts of the *Business Process Model and Notation* (BPMN) [16], we do not distinguish between "processes", "activities" and "sub-processes" but simply allow for composite processes. This does not contradict the BPMN approach and dramatically reduces the complexity of the execution engine because composition can be handled in a continuous way. Languages must implement a `CompositionResolver` as shown in Fig. 4 for the engine to be able to step through the levels of a process model.

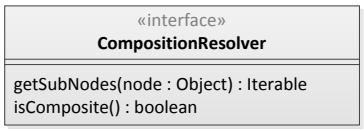

Figure 4. Interface to be implemented for the resolution of process compositions

Instead of transforming the process model into some runtime model, it is interpreted in place. For this purpose, execution pointers are maintained for every currently active process model node (cf. [15]). Like processes, these pointers may be composite and follow the structure of the process. Each execution pointer references the event trace of its node's child nodes.

In Fig. 5, process A is composed of the processes B and C. Process A is unfolded and the constraints for its children state that B may be activated in the current state. All events of a level are attached to the parent of this level. Therefore, the `Activation` event for B is attached to the `Execution` object of A. As B is now active, an `Execution` object is created for it as well.

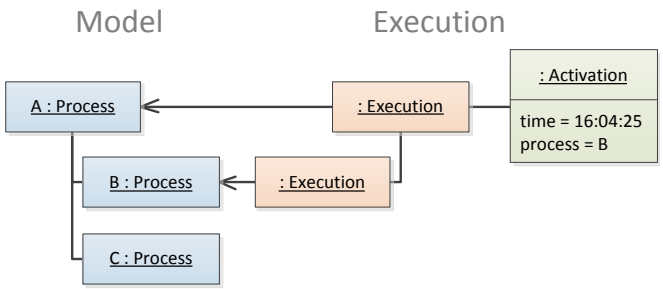

Figure 5. Example object graph during the execution of a composite process

Composition is a means of modularization. As a consequence, process constraints may only affect one level of a composition at once, i.e., one "sub-process" in terms of the BPMN. This restriction is comparable with the rule that, in BPMN, objects within a sub-process cannot be connected to objects outside of it through sequence flows [16].

D. *Traceability and Decision Support for Process Cases and Actions*

The scoring approach allows for before and after considerations concerning the rating of the process case. When the constraint interpretation is triggered, the process case may be assessed in its current state before appending a feasible next event. In Fig. 6, this state of an example process case consists of three events. It is found that it contradicts process constraint R_1 (which can only be a soft constraint as the events actually happened) so the case is currently in a discouraged state because it violates R_1. This information can be made available to business administrators. This scoring of the case's current state reveals the actual degree of deviation from the recommended process.

After the potentially next event e_4 is appended (b), the trace violates R_2. Therefore, the occurrence of event e_4 renders the case compliant with R_1 but conflicting with R_2 (c). This infor-

mation can be associated with the respective action so that its effect can be foreseen while the process is executed.

Up to the authors' knowledge, this level of decision support cannot be provided by any other process execution approach.

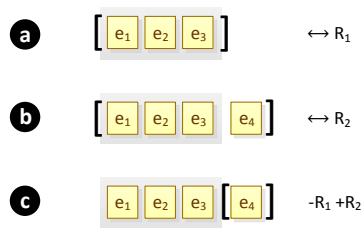

Figure 6. Scoring of a process case's current state (a), its potential next state (b) and the differential score of a potential next event (c)

E. *Extensibility and Adaptability*

The perspectives of a business process and their concrete shape may vary across different domains. A manufacturing process may focus on timing aspects like deadlines and cooling-off periods, while in a clinical process, data formats and dependencies may play an important role. In most cases, an according process modeling language will have to be chosen or created so that these aspects can be expressed. As long as the elements of this language can be expressed in terms of first-order logic, the approach at hand will support it. However, the language may only constrain facts that can be derived from the event trace. For this reason, the event type can be easily extended by language implementations. Thus, every detail necessary for the respective domain can be recorded in the trace and picked up by process constraints.

Fig. 7 shows an example extension for the manufacturing domain. Here it is necessary to constrain the part and tool used in a process step. The according `ManufacturingActivation` inherits from a `BaseActivation` which already allows for constraining the performer of a step.

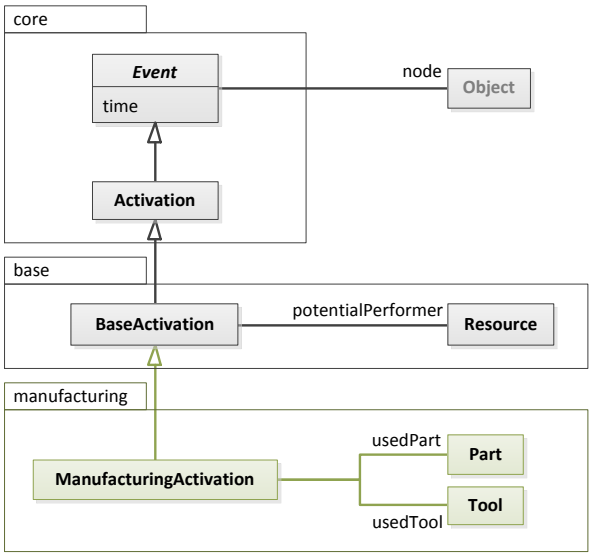

Figure 7. Class diagram of an event extension for the manufacturing domain

V. Prototypical Implementation and Integration

The task of finding all feasible next events within the boundaries of process constraints is essentially an optimization problem. During the past, basically two directions addressing this type of problem have evolved. On the one hand, mathematical approaches like, e.g., graph coloring or linear programming are applied. The *simplex* algorithm with its implementation in, e.g., *IBM® ILOG® CPLEX®* is a famous example. On the other hand, approaches based on the search in a state space are applied. Heuristic search algorithms, genetic algorithms and most logic programming systems fall into that category. An advantage of these search-based approaches over the mathematical ones is that they do not require the problem to be transformed into a special representation. The simplex method, e.g., requires the problem to be represented as a system of equations composed of matrices and vectors. However, due to the object-oriented paradigm, the data models to be optimized are usually represented as graphs of objects. Search-based approaches allow for these graphs to be optimized directly. Constraints can be expressed directly on the basis of these graphs as well. Therefore, the above concept is implemented on the bases of the search-based optimization framework *JBoss Drools Planner*. The framework uses meta-heuristic algorithms like tabu search [17] and simulated annealing [18] to find and evaluate solutions for NP-hard planning problems. The encountered solutions are rated on the basis of rules using an object-oriented variant of the Rete algorithm [19].

A. *Adapting the Drools Framework to Process Execution*

For the purpose of declarative business process execution, the combination of the current event trace and the possible next event is considered a *solution*. The event trace remains unchanged during the optimization procedure. Therefore, it is considered a *planning fact*. As the engine only proposes the activation of process steps, the possible next event is always an activation. It is varied during optimization which leads to alternative solutions. As a result, the `Activation` and its properties are annotated as the actual *planning entity* containing *planning variables*. Fig. 8 illustrates the relationship between the `Solution` and the `Activation` event.

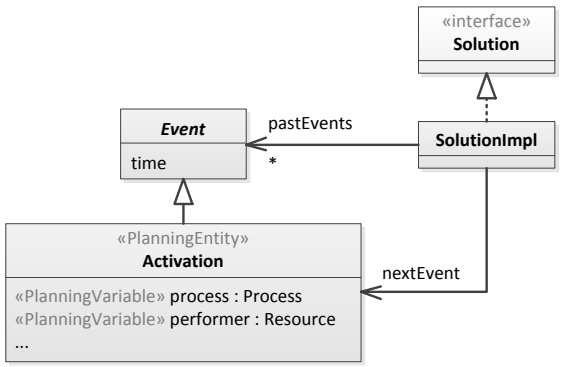

Figure 8. Class diagram showing the integration into the JBoss Drools Framework

The process constraints are expressed in the form of scoring rules in the framework's rule language named DRL [20]. This language allows for defining named rules (`rule` blocks) where the rule's conditional part (`when` clause) contains the negated constraint and the rule's consequence (`then` clause) states which type of constraint to violate. Condition expressions may bind objects to variables, check for the existence or non-existence of objects and may connect statements by logic operators. If for a process model a constraint *c* should hold (soft constraint), then the according DRL rule has the form

```
rule
  when
    ¬ c
  then
    breakSoftConstraint()
end
```

If a constraint *d* must hold (hard constraint) then the rule has the form

```
rule
  when
    ¬ d
  then
    breakHardConstraint()
end
```

As the `when` clause supports the full range of first order logic expressions, the approach supports highly expressive process modeling languages. Rules may constrain the existence and order of steps, the performing agents, the incorporated data and the utilized tools. Please note that DRL itself is not intended to be an actual process constraint language. An end-user language must offer a coarser granularity and must be based on patterns.

By default, the Drools solver only returns the "best" solution where the definition of "best" must be configured. Returning only one solution would imply that in each state of the process case only the most recommended next step is proposed. Of course, all feasible next steps should be offered so the default behavior of the solver must be changed accordingly. This is achieved by implementing a custom `BestSolutionRecaller` strategy which collects all of the feasible solutions encountered during the evaluation procedure.

B. Task Management

As it is common practice, the core execution engine is separated from the management of human tasks. In this context, the *WS-HumanTask* specification [21] defines an interface for querying and manipulating human tasks. However, it does not allow for scores to be transported. This is necessary because, in contrast to classic engines, e.g., people assignments are rated (variously recommended). As a result, the interface must be extended so that tasks, potential owners and data items may carry scores. Fig. 9 shows that the task management interface is inspired by the WS-HumanTask API but augmented by scoring. It is actually possible to create an adapter for connecting the engine to standard WS-HumanTask clients.

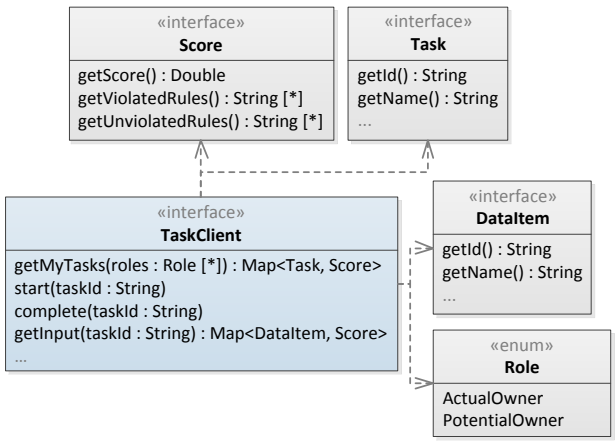

Figure 9. Scoring-enabled task management interface for task clients

A score consists of the actual soft score value, references to the violated process constraints (rules) and references to the "unviolated" constraints (rules that the process complies with again after the action). Instead of returning a list of tasks, a query maps actually or potentially owned tasks to scores. Also the input data items are scored as it may be recommended to use a certain document instead of another one.

The score-enabled task management interface facilitates task clients that are capable of transporting more information than the classic WS-HumanTask clients and therefore supports more informed decisions of the process participants.

VI. Running Example

In the following, a real world example illustrates the steps from a business process model in the form of process constraints to scoring rules and outlines possible execution paths. The example originates from the engineering domain where employees elaborate CNC models which are executed by a CNC lathe. The process cannot be foreseen and defined completely. However the following restrictions are made:

- "The CNC model should not be created by a trainee."
- "However, if the CNC model has been created by a trainee, then it must be reviewed by his supervisor before it can be submitted to the CNC lathe."

The constraints reflected by this example intersect temporal and causal dependencies ("if" and "before") with organizational ones ("by a trainee" and "by his supervisor") and make use of different modalities ("should" and "must"). The following paragraphs illustrate how these statements may be translated to closed formal constraints and interpreted by the execution engine described above.

A. Formal Constraints

The above prose statements have to be transformed into first-order logic expressions referring to the event trace. It is assumed that there are at least two processes "Create Model" and "Review Model" and that there is a group called "Trainees". The first statement claims that there should be no activation of the process "Create Model" with a potential performer who is member of the "Trainees" group:

$$\nexists a \,(\text{Activation}(\text{Create Model}, a) \wedge \text{memberOf}(a, \text{Trainees})) \quad (4)$$

The second statement claims that if there is a completion of "Create Model" by a member of the "Trainees" group, then there must be an activation of "Review Model" by the supervisor of this agent:

$$\exists a, s \,(\text{Completion}(\text{Create Model}, a) \wedge \text{memberOf}(a, \text{Trainees}) \rightarrow \text{Activation}(\text{Review Model}, s) \wedge \text{supervisorOf}(s, a)) \quad (5)$$

B. Scoring Rules

As mentioned above, the purpose of scoring rules is to punish the negation of the constraint. So the expressions from (4) and (5) are negated and expressed as DRL rules:

```
rule "Model not created by trainee"
  when
    $p : Process(id == "Create Model") and
    $t : Group(id == "Trainees") and
    $a : Agent(memberOf contains $t) and
    exists Activation(process == $p,
      performer == $a)
  then
    breakSoftConstraint()
end

rule "If model created by trainee then
reviewed by supervisor"
  when
    $p1 : Process(id == "Create Model") and
    $t : Group(id == "Trainees") and
    $a : Agent(memberOf contains $t) and
    exists Completion(process == $p1,
      performer == $a, $c : time) and
    $p2 : Process(id == "Review Model") and
    $s : Agent(supervisorOf contains $a) and
    not exists Activation(process == $p2,
      performer == $s, time > $c)
  then
    breakHardConstraint()
end
```

Expression (4) results in the rule "Model not created by trainee". It binds the process called "Create Model", the group called "Trainees" and all the agents that are member of this group. If there exists an activation for the process and one of these agents then the according solution violates a soft constraint. A reference to the rule together with the variables' bindings is associated with the solution for traceability.

Expression (5) can be expressed as the rule "If model created by trainee then re-viewed by supervisor". The rule binds the process "Create Model" and "Review Model", the members of "Trainees" and their supervisors. If now there is a completion for "Create Model" by a trainee and no activation for "Review Model" for his or her supervisor then this solution violates a hard constraint.

The above rules demonstrate the expressivity and lowest granularity of the approach. They are at a very low level of abstraction and are not meant to be an actual means of modeling business processes. Instead, an end-user process modeling language is intended to be a set of rule templates possibly together with a graphical notation and validity rules.

C. *Simulation of Possible Execution Paths*

For this small example, possible traces may be easily inferred as shown in Fig. 10. We assume that there are the agents "Alice" and "Bob" while "Alice" is the supervisor of "Bob" who is a trainee. Initially, "Create Model" is activated with both "Alice" and "Bob" as potential performers. However, the activation for "Bob" contradicts the first constraint so the event is marked discouraged and a reference to the violated constraint is attached to it. When a work item is generated, it may be annotated accordingly. If "Alice" completes the task then "Submit Model" is directly activated afterwards.

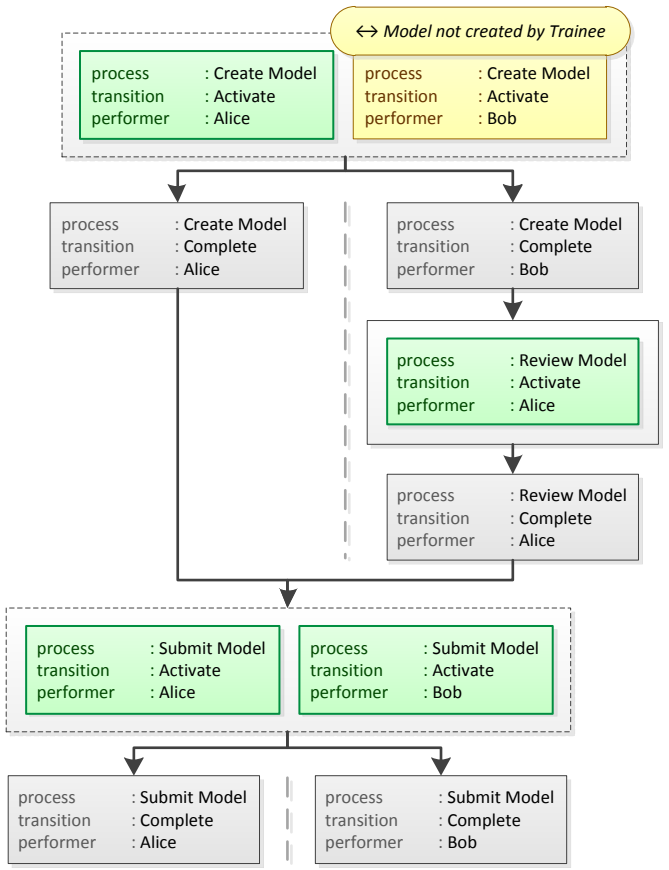

Figure 10. Exemplary paths when executing the example process model

However, if "Bob" creates the model only "Review Model" is activated for "Alice" because the second constraint claims that a review of the trainee's supervisor has to follow the creation by a trainee.

Please note that the illustration in Fig. 10 is not complete. There are a lot more possible paths resulting from these few restrictions.

VII. Conclusion, Limitations and Future Work

Declarative business process modeling turns out to be best suited for the description of so-called flexible business processes comprising a high number of possible paths. Current engines for the execution of such processes lack certain crucial characteristics as they

- do not support constraints to span multiple perspectives of a process or
- may not explain and justify the proposed actions or
- do not scale sufficiently to handle real-world business processes.

The presented approach executes declarative process models based on first-order logic constraints that may span multiple perspectives of a business process like, e.g., incorporated data, agents performing the work and utilized tools. It dynamically interprets the model and therefore scales better than approaches that completely pre-calculate the possible paths. Due to its architecture, not recommended actions may be traced back and explained by the original process constraints. Thereby, the approach addresses the above issues and promises to enable practicable declarative business process management.

As mentioned above, the declarative execution engine is part of the modular multi-paradigm platform *AI4 Process Navigation*. Therefore, one objective is to further unify these two architectures and to extract common elements so that, in the end, the actual imperative and the declarative execution cores only contain functionality that is specific to the respective paradigm.

The described architecture shows that the state of the system only changes when it is triggered from outside, e.g., by a complete event. Accordingly, the progression of time may not trigger a state change. It remains to be examined to what extent this impedes constraints based on absolute time like, e.g., cooling-off periods or deadlines. As a result, a support for time-triggered execution remains to be evaluated.

We argue that the dynamic interpretation of declarative process models improves scalability. Up to now, this is a theoretical advantage that needs to be proven by concretely demonstrable results. The according benchmarks and comparisons are to be developed in the future.

As Edsger Dijkstra already realized, "our powers to visualize processes evolving in time are relatively poorly developed" [22]. Actually, rule-based descriptions of business processes are known to suffer from understandability issues [3]. One way of addressing this problem is to continuously simulate the execution of a process model. Therefore, a further objective is to develop a framework for the stepwise simulation of declarative process models so that their behavior may be fully understood.

REFERENCES

[1] M. Pešić, H. Schonenberg, and W. M. P. van der Aalst, "Declarative Workflow," in *Modern Business Process Automation: YAWL and its Support Environment*, A. H. M. ter Hofstede, *et al.*, Eds., ed: Springer, 2010, pp. 175-201.

[2] S. Sadiq, W. Sadiq, and M. Orlowska, "Pockets of Flexibility in Workflow Specification," in *20th International Conference on Conceptual Modeling (ER'2001)*, Yokohama, Japan, 2001.

[3] D. Fahland, D. Lübke, J. Mendling, H. Reijers, B. Weber, M. Weidlich, et al., "Declarative versus Imperative Process Modeling Languages: The Issue of Understandability," in *Enterprise, Business-Process and Information Systems Modeling (10th International Workshop, BPMDS 2009, and 14th International Conference, EMMSAD 2009, held at CAiSE 2009)*, Amsterdam, The Netherlands, 2009, pp. 353-366.

[4] P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw, "Freeflow: Mediating Between Representation and Action in Workflow Systems," in *ACM Conference on Computer Supported Cooperative Work (CSCW '96)*, Boston, MA, US, 1996.

[5] W. M. P. van der Aalst, "TomTom for Business Process Management (TomTom4BPM)," in *21st International Conference on Advanced Information Systems Engineering (CAiSE 2009)*, Amsterdam, NL, 2009, pp. 2-5.

[6] E. Turban, R. Sharda, and D. Delen, *Decision Support and Business Intelligence Systems*, 9 ed.: Prentice Hall, 2010.

[7] S. Jablonski and C. Bußler, *Workflow Management: Modeling Concepts, Architecture and Implementation*. London: Thomson, 1996.

[8] M. Igler, M. Faerber, M. Zeising, and S. Jablonski, "Modeling and Planning Collaboration in Process Management Systems using Organizational Constraints," in *The 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2010)*, Chicago, IL, USA, 2010, pp. 1-10.

[9] M. Pešić, "Constraint-Based Workflow Management Systems: Shifting Control to Users," Dissertation, Technische Universiteit Eindhoven, 2006.

[10] S. Sadiq, M. Orlowska, and W. Sadiq, "Specification and Validation of Process Constraints for Flexible Workflows," *Information Systems*, vol. 30, pp. 349-378, 2005.

[11] J. Wainer and F. Bezerra, "Constraint-based Flexible Workflows," in *Groupware: Design, Implementation and Use*, Autrans, FR, 2003, pp. 151-158.

[12] S. Goedertier, R. Haesen, and J. Vanthienen, "Rule-based Business Process Modelling and Enactment," *Business Process Integration and Management*, vol. 3, pp. 194-207, 2008.

[13] M. Igler, S. Jablonski, M. Zeising, and P. Moura, "ESProNa: Constraint-based Declarative Business Process Modeling," in *14th IEEE International Enterprise Distributed Object Computing Conference (EDOCW 2010)*, Vitória, ES, BR, 2010, pp. 91-98.

[14] S. Jablonski, M. Igler, and C. Günther, "Supporting Collaborative Work through Flexible Process Execution," in *5th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2009)*, Crystal City, Washington D.C., USA, 2009.

[15] T. Baeyens and M. V. Faura. (2007). *The Process Virtual Machine*. Available: http://docs.jboss.com/jbpm/pvm/article/

[16] I. Object Management Group, "Business Process Model and Notation (BPMN) Version 2.0," ed, 2011.

[17] F. Glover, "Tabu Search - Part I," *ORSA Journal on Computing*, vol. 1, pp. 190-206, 1989.

[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, 1983.

[19] C. Forgy, "On the Efficient Implementation of Production Systems," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, PA, US, 1979.

[20] JBoss. (2012, 05/07/2012). *Drools Expert User Guide Version 5.4.0.Final*. Available: http://docs.jboss.org/drools/release/5.4.0.Final/drools-expert-docs/html/

[21] OASIS, "Web Services Human Task (WS-HumanTask), Version 1.0," ed, 2007.

[22] E. W. Dijkstra, "Go To Statement Considered Harmful," *Communications of the ACM*, vol. 11, pp. 147-148, 1968.