Towards an Implementation of Data and Resource Patterns in Constraint-based Process Models

Stefan Schönig, Lars Ackermann and Stefan Jablonski
Institute for Computer Science, University of Bayreuth, Bayreuth, Germany

Keywords: Constraint-based Processes, Flexible Processes, Workflow Patterns, Process Execution.

Abstract: A Process-Aware Information System (PAIS) is a system that executes processes involving people, applications, and data on the basis of process models. Two representations for processes can be distinguished: procedural models prescribe exactly the execution order of process steps. Declarative process models allow flexible process executions that are restricted by constraints. Especially in application areas of knowledge driven processes, this flexibility is required. Foundations of declarative approaches have been extensively discussed in research. From a practitioner's point of view, however, an open question still remains: is it possible to implement established functionality in contemporary declarative PAIS, especially data and resource handling? In this paper, we tackle this open research question by introducing the declarative process modelling and execution framework DPIL that covers resource and data modelling. Expressiveness and functionality of the framework are evaluated by means of the well-known Workflow Data and Resource Patterns.

1 INTRODUCTION

A Process-Aware Information System (PAIS) is a software system that manages and executes business processes involving people, applications, and data on the basis of process models (Reichert and Weber, 2012). Processes can be modelled using a variety of languages and different concepts. State of the art technology distinguishes between two principle approaches: procedural concepts prescribe exactly the execution order of process steps. Declarative, i.e., constraint-based processes on the other hand allow flexible process executions that are restricted by constraints (Fahland et al., 2009). Especially in application areas of knowledge driven processes this flexibility is required since the exact flow of activities cannot be fully determined at design time (Reichert and Weber, 2012). Here, the execution sequence heavily depends on human participants, their decisions and expert knowledge (Fahland et al., 2009). Recent research has shown that declarative approaches are best suited for supporting flexible, knowledge driven processes (Fahland et al., 2009; Vaculín et al., 2011).

Independent from the modelling paradigm processes can be seen from different perspectives (Jablonski and Bussler, 1996). The behavioural perspective describes activities and their execution ordering, e.g., task sequences or parallelism. The data perspective deals with process data and documents. Business documents and other objects which are used within activities as well as local variables of the process, may reflect pre- and post-conditions of activity execution. Typically, process data is passed into and out of applications through interfaces, allowing manipulation of the data. The resource perspective manages the involvement of resources in processes, e.g., it provides an anchor to the process in the form of human roles responsible for activities.

Theoretical foundations of declarative approaches have been extensively discussed in research with a strong focus on behavioural aspects (Montali, 2010; Pesic, 2008) and conceptually extensions for data (Westergaard and Maggi, 2012; Montali et al., 2013; Burattin et al., 2015). From a practitioner's point of view, however, an open question still remains (Reijers et al., 2013): is it possible to implement established functionality in contemporary declarative PAIS, in particular data and resource handling? Answering this question is essential for using declarative approaches in practice.

In this paper, we tackle this open research question by introducing the declarative process modelling and execution framework *DPIL* that covers in particular resource and data modelling. Expressiveness and functionality of the framework are evaluated by means of the well-known *Workflow Data and Re-*

Schönig, S., Ackermann, L. and Jablonski, S.
Towards an Implementation of Data and Resource Patterns in Constraint-based Process Models.
DOI: 10.5220/0006533502710278
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 271-278
ISBN: 978-989-758-283-7

source Patterns[1], a systematically specified catalogue of desired functionality of PAIS. The set of workflow patterns covers recurring requirements focusing on data interaction (Russell et al., 2005a) and resource involvement (Russell et al., 2005b). We give concrete model excerpts that implement frequently needed functionality. Examplary real-life processes that include these patterns can be executed with the *DPIL Navigator* at http://navigator.kppq.de. The contribution of this work is to show that existing declarative PAIS are ready to use for practical applications. The remainder of this paper is structured as follows: in Section 2 we introduce the syntax and grammar of the DPIL modelling and execution framework. Section 3 describes the implementation of Data and Resource Patterns in DPIL. In Section 4 we discuss related work and the paper is concluded in Section 5.

2 MODELLING WITH DPIL

In this section, we introduce the framework we use to implement the workflow patterns. The *Declarative Process Intermediate Language (DPIL)* is a multi perspective and multi modal process modelling language on a textual basis. Unlike other declarative languages it allows for representing several perspectives, namely, behaviour, data and resources and their crosscutting relations. It is multi modal, meaning that both mandatory and recommended actions are specified. The DPIL framework offers a rich toolset for modelling and executing declarative process models: *(i)* DPIL models can be defined using a textual editor the *DPIL Modeller*. The modeling tool supports users through syntactic, semantic and qualitative model analysis based on the Xtext framework[2]. DPIL models can be *(ii)* executed by the declarative execution engine, the *DPIL Navigator*. Both tools have been published in demo papers (Schönig and Zeising, 2015; Schönig et al., 2017). DPIL allows for defining reusable templates (*macros*) in order to keep the resulting model concise. Macros can be defined to the modellers' needs using all concepts of DPIL.

2.1 The DPIL Language

Hereafter a concrete syntax is described that is used to represent a DPIL model textually. The concrete syntax is described using the Antlr conform extended Backus-Naur form (EBNF).

[1]Information at http://www.workflowpatterns.com
[2]http://www.eclipse.org/Xtext/

2.1.1 Models

In the model head the used identities, groups and relation types from the organizational model are enumerated to make them available and referable by their names. Additionally connection properties to CMIS compatible ECM systems[3] can be specified to describe source and target of data objects. To avoid redundancy one can additionally define macros or global rules in this section of the model. Consequently, a model might lack any process information but instead can be used to form a library model by just enumerating elements of the organizational model, defines macros or connections to ECM systems. Models can reference other models and, thus, can make use of, for instance, this library model.

```
Model: Identity* Group* RelationType* Repository* Macro*
       GlobalRule* Process?
Identity: 'identity' ID
Group: 'group' ID
RelationType: 'relationtype' ID
Repository: 'repository' ID '{'
            'url' STRING
            'user' STRING
            'password' STRING
            'id' STRING '}'
```

2.1.2 Structural Elements

A *process* can consist of activities, data objects and process rules. Activities and data objects must have an identifier (*ID*) and can be named. An activity is, in turn, a process, a human *task* or an *operation*. An operation omits the name and only consists of an identifier and a body. Additionally it is possible to assign actual arguments to the operation's dummy arguments as well as a data object to store the return value (*to*). Data objects can either be variables or documents. Both can be multi-valued (*collection*) and documents must be further specified by a query for the *default* value and a repository connection identifier (*at*).

```
Process: 'process' ID STRING? '{'
          Activity*
          DataObject*
          ProcessRule* '}'
Activity: Process | Task | Operation
          Task: 'task' ID STRING?
          Operation: 'operation' ID STRING
          ('{' Parameter (',' Parameter)* '}')?
          ('to' ID)
Parameter: ID ID
DataObject: Variable | Document
Variable: 'variable' 'collection'? ID STRING?
Document: 'document' 'collection'? ID STRING?
```

[3]CMIS = Content Management Interoperability Services, ECM = Enterprise Content Management

```
        ('default' STRING)?
        'at' ID
```

2.1.3 Rules

Below the concrete syntax for structural elements is complemented by the rule syntax. To focus on a description that makes the rule part of the language usable we omit a detailed discussion of the abstract syntax. Global rules can be either mandatory (*ensure*) or just recommendations (*advise*). A process rule can be supplementary a *milestone*. Rules can have an identifier and a description which have to be separated from the body using a colon. Macros must have an identifier and can make use of dummy parameters. The macro's body is separated from the head using the *iff* keyword.

```
GlobalRule: GlobalRuleType (ID? STRING? ':')? Expression
GlobalRuleType: 'ensure' | 'advise'
ProcessRule: ProcessRuleType (ID? STRING? ':')? Expression
ProcessRuleType: 'ensure' | 'advise' | 'milestone'
Macro: ID ('(' ID (',' ID)* ')')? 'iff' Expression
```

According to the metamodel a rule expression is a tree structure whereby its nodes are either unary or binary expressions. This nesting has to be considered in the grammar draft. For all operators the grammar realises an infix notation, i.e., the operator is always located between its operands. Furthermore, the grammars stipulates an operator priority ranking which, for instance, assigns the highest priority to parentheses, negations etc. and the lowest to the implication. The universal quantifier (*forall*) stipulates, based on the objects matched by the first pattern, all further patterns. A predicate reference consists of the ID of the referenced predicate (macro or rule) and a list of its actual parameters. Object selectors can select objects either using their identifier (*ObjectReference*) or using any other arbitrary property (*ObjectConstraint*). The selection result can be assigned to a variable using the colon operator and the variable identifier.

```
Expression: Implies
Implies: Or ('implies' Or)*
Or: And ('or' And)*
And: Unary ('and' Unary)*
Unary: '(' Expression ')'
       | 'not' Unary
       | 'exists' Unary
       | Forall
       | PredicateReference
       | ObjectSelector
Forall: 'forall' '(' ObjectSelector ObjectSelector+ ')'
PredicateReference: ID ('(' LiteralOrReference
(',' LiteralOrReference)* ')')?
ObjectSelector: ObjectConstraint | ObjectReference
ObjectReference: Type ID (':' ID)?
ObjectConstraint: Type ('(' ConstraintExpression ')')?
(':' ID)?
Type: 'task' | 'operation' | 'process' | 'start' |
```

To select objects by properties not by their identifier but instead by other properties one has to provide an ObjectConstraint with a corresponding *Constraint-Expression*. This again is formed by a tree structure of unary and binary subexpressions. Operators are again realised using the infix notation with the same operator prioritization like the operators of rule expressions. The properties may either be bound to variables (*PropertyBinding*). or be compared to other variables or constants (*PropertyRestriction*).

```
ConstraintExpression: ConstraintOr
ConstraintOr: ConstraintAnd ('or' ConstraintAnd)*
ConstraintAnd: ConstraintUnary ('and' ConstraintUnary)*
ConstraintUnary: PropertyBinding
               | PropertyRestriction
               | '(' ConstraintExpression ')'
PropertyBinding: PropertyReference ':' ID
PropertyReference: PropertyKey RQID?
PropertyKey: 'this' | 'of' | 'by' |
PropertyRestriction: PropertyReference Operator?
LiteralOrReference
Operator: '=' | '!=' |
LiteralOrReference: STRING | NUMBER | ID
```

2.2 Execution

Some of the elements of a DPIL process model undergo a life cycle composed of events that is managed by the engine. A human task, e.g., can be started and completed while a data object can be read or written. The current state of a process is then the series of past events. Besides the static elements like human tasks and data objects, a process model may specify rules constraining that series of events. It may, e.g., claim that some data object may only be written after some task has been started. These rules may be hard to reflect, e.g., the legal framework or soft to reflect, e.g., good practice. When the model is executed, the engine simulates one event ahead for every model element and evaluates the resulting series of events on the basis of the rules. Each simulated event that does not violate any hard rule is related to an action that engine interprets immediately. A simulated start of a task by a certain participant, e.g., is interpreted as the assignment of this task to the participant. If the start event violates a soft rule, the action is marked as not recommended.

After generating the events the engine has all events for one process instance that might occur in this instance available. The generation of these events does not consider the modelled process rules. Thus the innovative core of the execution is first to evaluate these events based on the process rules and to narrow them afterwards. For this purpose each possible event is combined with the former course of events of the current instance and for each of these combined courses the engine decides to what extent it conforms to the process rules. Hence, it must be decided whether a simulated event and, consequently, an enabled action in the process is forbidden, non-recommended, neutral or recommended. The engine itself imposes only few requirements on a rule-based model. From the perspective of the engine a process model M consists of model elements E and rules R, i.e., $M = (E,R)$. Model elements are entities for which events might happen like, for instance, activities and data objects. A rule r is a logic proposition that can be fulfilled or violated. The set of rules R of a process model can be separated into hard rules R_H, soft rules R_S and milestones R_M, i.e., $R = R_H \cup R_S \cup R_M$. A process instance I is a set of materialized events within one process. For the engine these events do not have to be ordered. An instance is considered to be completed as soon as all defined milestones are fulfilled. If no milestones are defined it cannot be determined when the corresponding process is completed. In this case, for the engine, a process instance can never be completed and is aborted with the termination of the DPIL Navigator platform. In order to simplify the subsequent definitions the change δ of a set L to a different set R is defined. A change is a tuple consisting of the set of removed and the set of added elements, i.e., $\delta(L,R) = (L \setminus R, R \setminus L)$. For the change of the set $\{a,b\}$ to the set $\{b,c\}$ one has: $\delta(\{a,b\},\{b,c\}) = (\{a\},\{c\})$.

2.2.1 Event Evaluation

An evaluation ρ is, in general, a set of violated rules of R. Hence, it always pertains: $\rho \in R$. The event evaluation $\rho(I \cup e)$ is the evaluation of an instance I after the occurrence of an event e. The event is considered to be forbidden if an evaluation after its occurrence overlaps with the set of hard rules R_H. It is considered to be permitted if the intersection is empty.

$$\begin{aligned} forbidden(e) &\leftarrow \rho(I \cup e) \cap R_H \neq \emptyset \\ allowed(e) &\leftarrow \rho(I \cup e) \cap R_H = \emptyset \end{aligned} \tag{1}$$

An event is treated as final if milestones are defined and the evaluation of the event does not overlap with them. Thus, with the occurrence of the event all milestones are fulfilled.

$$final(e) \leftarrow R_M \neq \emptyset \wedge \rho(I \cup e) \cap R_M = \emptyset \tag{2}$$

2.2.2 Change of Instance Evaluation

The instance evaluation change $\Delta\rho_I$ is the change of the evaluation of an instance I with the occurrence of an event e. Hence, it describes the repercussions of the event on the instance.

$$\Delta\rho_I(e) = \delta(\rho(I), \rho(I \cup e)) \tag{3}$$

An event is considered to be neutral if the instance evaluation change is empty, i.e. the event does not have any repercussion on the instance.

$$neutral(e) \leftarrow \Delta\rho_I(e) = (\emptyset, \emptyset) \tag{4}$$

An event e is treated as recommended, if it is permitted and if it leads to an abolition of previous violations P of soft rules or milestones and no other rules are violated in consequence of this event.

$$\begin{aligned} recommended(e) \leftarrow \Delta\rho_I(e) = \\ (P, \emptyset) \wedge P \cap (R_S \cup R_M) \neq \emptyset \end{aligned} \tag{5}$$

Conversely it is non-recommended if it is permitted but leads to new violations Q of soft rules or milestones.

$$\begin{aligned} notRecommended(e) \leftarrow \Delta\rho_I(e) = \\ (P, Q) \wedge Q \cap (R_S \cup R_M) \neq \emptyset \end{aligned} \tag{6}$$

2.2.3 Change of Event Evaluation

The event evaluation change ΔV_e is the change of the event evaluation regarding the previous execution step:

$$\begin{aligned} \Delta\rho_e(e) = \delta(\rho(I \cup e)_{t-1}, \rho(I \cup e)_t) \; with \\ \rho(I \cup e)_t = \emptyset \; for \; t < 0 \end{aligned} \tag{7}$$

Thus an event is considered to be changed if its event evaluation change set is not empty or if the event denotes the first time step.

$$changed(e) \leftarrow \Delta\rho_e(e) \neq (\emptyset, \emptyset) \vee t = 0 \tag{8}$$

An event is considered to be newly permitted if it was forbidden in the previous time step and if it is permitted in the current time step. Additionally it is treated as newly permitted if it is permitted in the current time step and if it denotes the first time step and, by association, the first event after the start of the process instance. In this case no previous events exist.

$$\begin{aligned} newlyAllowed(e) \leftarrow \Delta\rho_e(e) = \\ (P, Q) \wedge Q \cap R_H = \emptyset \wedge (P \cap R_H \neq \emptyset) \vee t = 0 \end{aligned} \tag{9}$$

An event is considered to be newly forbidden if it was permitted in the previous time step but is forbidden in the current one. Like in the previous paragraph it is also treated as newly forbidden if it is forbidden in the current time step and if it is the first time step.

$$\begin{aligned} newlyForbidden(e) \leftarrow \Delta\rho_e(e) = \\ (P, Q) \wedge Q \cap R_H \neq \emptyset \wedge (P \cap R_H = \emptyset) \vee t = 0 \end{aligned} \tag{10}$$

3 WORKFLOW PATTERNS

In this section, we evaluate the expressiveness of the framework by describing how Data and Resource Patterns can be implemented in DPIL. Therefore, we provide a brief description of the patterns as well as DPIL model excerpts that reflect the desired behaviour. The set of control flow patterns (van Der Aalst et al., 2003) mainly describes functionality stemming from classical procedural PAIS, i.e., traditional elements of procedural languages like BPMN. Therefore, they are only conditionally applicable for declarative process modelling and are not considered in this paper.

3.1 Workflow Data Patterns

The way in which process data is structured and included in the process is described in the workflow data patterns (Russell et al., 2005a). The patterns are structured in visibility of data, interaction with data, data transfer and data-based routing.

3.1.1 Visibility

A data variable is visible within the selected process instance (WDP5) and within the corresponding subprocesses (WDP2). In order to limit the visibility of a data object to certain activities (WDP1), read and write events can be constrained in the following way.

```
local(task, object) iff
read(of object at :tr) implies start(of task at < tr at :ts)
and not complete(of task at > ts at < tr)

task CalculateFlightPath
variable WorkingTrajectory
ensure local (CalculateFlightPath, WorkingTrajectory)
```

This way the data object *WorkingTrajectory* is only visible within the activity *CalculateFlightPath*. The lifespan of variables and its values, however, is equal to the duration of the process instance. Access to data objects can be dependent on the corresponding process instance (WDP6). The following code shows how it is possible to access different collections of documents in activity *AccessFolder*, depending on which folder has been selected in *SelectFolder*:

```
task SelectFolder
task AccessFolder

document collection FolA default "/A" at Local
document collection FolB default "/B" at Local

variable FolderName

ensure produces (SelectFolder, FolderName)
ensure sequence(SelectFolder, AccessFolder)
ensure read(of FolA) implies write(of FolderName value "A")
ensure read(of FolB) implies write(of FolderName value "B")
```

In order to make environment data accessible by the system, it has to be read by means of an operation method and written in a variable. This is explained by the following model that loads the current temperature to a variable *Temperature*.

```
operation GetTemperature "http://service.org/temperature"
        to Temperature
task CheckTemperature
variable Temperature
ensure start(of CheckTemperature) implies
return(of GetTemperature)
```

3.1.2 Interaction

Data objects are visible for all activities of the case. Their values can be communicated between these activities as well (WDP9). Since data objects are also visible for all subprocesses of a process case, values can also be passed to them implicitely (WDP10) and reused (WDP11) by them. Since multi instance activities are not supported in DPIL, it is also not possible to pass data values between them (WDP12, 13). Data interaction between several parallel process cases (WDP14) can be implemented by means of externally managed documents, i.e., a CMIS compatible document management system. As shown above, environment data can be passed (WDP15) and queried (WDP16) by means of operations, i.e., script calls. Data objects can be implicitely received during activity execution (WDP17) by means of externally managed and changed documents. This way data objects can also be returned (WDP18). Since data objects are visible for all tasks of a process this also holds for a complete instance (WDP21, 22).

In case that only externally managed documents are used within the process, data values of corresponding process cases are synchronized with the connected document management system. The states of process cases is therefore visible to the environment. Furthermore, data values can be pushed from cases to the environment by means of a corresponding operation call at the end of a case (WDP19). In any case data can be passed from the environment to running cases by means of externally managed documents (WDP20). The DPIL Navigator supports process oriented indicators like the number of completed cases of a process. These numbers can be transfered to the environment regularly (WDP23) or offer them upon request (WDP26).

3.1.3 Transfer

Access to data objects generally happens by reference (WDP30). Locking mechanisms can be implemented by means of constraints on write and read events (WDP31). The transformation of values (WDP32, 33) can be handled by means of service operations:

```
operation FromKmhToMph "http://service.org/kmhToMph"
          with SpeedKmh to SpeedMph
task ReviewSpeed
variable SpeedKmh
variable SpeedMph
ensure start(of ReviewSpeed) implies return(of FromKmhToMph)
ensure invoke(of FromKmhToMph) implies write(of SpeedKmh)
```

3.1.4 Data-based Routing

Pre- and postconditions for activities can be implemented by constraining start and compete events of activities. Events can depend on the existence of a value or on a concrete value (WDP34-37). Both patterns are highlighted in the following example:

```
consumes(t, d) iff start(of t) implies write(of d)
produces(t, d) iff complete(of t) implies write(of d)

task RocketInitiation

variable Countdown
variable IgnitionData

ensure consumes(RocketInitiation, Countdown)
ensure start(of RocketInitiation) implies
       write(of Countdown value 2)
ensure produces(RocketInitiation, IgnitionData)
```

Activities can be triggered by the environment (WDP38):

```
task Shutdown
operation MonitorAlarm "org.example.Alarm.monitor()" to Alarm
variable Alarm
ensure start(of Shutdown) implies return(of MonitorAlarm)
```

Activities can also be triggered by certain data values (WDP39):

```
task RebalancePortfolio
variable LoanMargin
ensure start(of RebalancePortfolio) implies
       write(of LoanMargin value > 0.85)
```

Data-based routing (WDP40) in declarative models corresponds to an exclusive split (WCP4) or a multi choice branch (WCP6), respectively.

3.2 Workflow Resource Patterns

Patterns concerning the organizational aspects of a process are summarized in (Russell et al., 2005b) where participants are referred to as *resources*. They are divided into patterns that refer to the design phase of the process (creation), to the systems perspective (push), to the participants perspective (pull), to exceptional situations (detour), to event- or context-based execution (auto-start), to visibility and to multiple participants working on the same task. DPIL assumes a simple organizational metamodel where identities and groups can be interconnected by relations of a certain relation type. The information is taken from a central organizational model like, e.g., an LDAP service and is only referenced within the DPIL process model. Human participation in a DPIL process is implemented using the task type of activity. From the engines viewpoint, a task may be started and completed. The task management service of the DPIL platform extends the life cycle of a task by reserve/release and suspend/resume states.

The simplest method of distribution is the direct allocation of a participant to a task at design time (WRP1). Besides this, tasks may be distributed on the basis of roles (WRP2), capabilities (WRP8) or organizational relationships (WRP10). The following example process contains each of these patterns:

```
use identity Fred
use group Manager
use group Engineer
use relationtype hasRole
use relationtype hasJob
use relationtype isManagerOf

direct(t, i) iff start(of t) implies start(of t by i)
role(t, r) iff start(of t by :i)
    implies relation(subject i predicate hasRole object r)

process Organisation {
    task FixBentley
    task ApproveTravelRequisition
    task AirframeExamination
    task ClaimExpenditure
    task AuthoriseExpenditure

    advise direct(FixBentley, Fred)
    advise role(ApproveTravelRequisition, Manager)

    advise start(of AirframeExamination) implies
        start(of AirframeExamination
              by :i by.servicingExperienceInYears >= 10)
        and relation(subject i predicate hasJob object Engineer)

    advise start(of ClaimExpenditure by :claimer)
        and start(of AuthoriseExpenditure by :authoriser)
        implies relation(subject authoriser
                         predicate isManagerOf
                         object claimer)
}
```

The above model references the identity *Fred*, two groups and three relation types. The task *FixBentley* should only be performed by *Fred* (WRP1). The rule is soft (type *advise*) so that the assignment to Fred is recommended but not mandatory, i.e., other participants are allowed to perform the task but they are advised not to do so. The task *ApproveTravelRequisition* should be performed by a participant having the role of a *Manager* (WRP2). The *AirframeExamination* should be performed by an *Engineer* having at least ten years of servicing experience (WRP8). Finally, AuthoriseExpenditure should be performed by the manager of the participant who has performed *ClaimExpenditure* (WRP10). DPIL supports the separation (WRP5) and the binding (WRP7) of duties for activities. The following model shows how different tasks are performed by the same and by different resources:

```
separate(a, b) iff
start(of b by :ib) implies start(of a by != ib)
retain(a, b) iff
start(of b by :ib) implies start(of a by ib)

process Organisation {
    task UmpireMatch
    task PrepareMatchReport
    task PrepareCheque
    task CountersignCheque

    advise retain(UmpireMatch, PrepareMatchReport)
    advise separate(PrepareCheque, CountersignCheque)
}
```

An assignment can be deferred (WRP3) by assigning the concrete identity that should perform a task at runtime through the value of a certain data object:

```
task AssessDamage
variable Performer

advise start(of AssessDamage) implies
       variablewrite(of Performer value :p)
       and start(of AssessDamage by.id p)
```

DPIL does not offer any means of referencing earlier process instances (WRP9) and is not able to perform any scheduling (WRP15-17). Activities can be offered without obligation of a single (WRP12) or of several resources (WRP13) by modelling assignment constraints as soft rules (advise). A mandatory assignment to a certain resource (WRP14) can be specified by means of hard rules (ensure). Participants can reserve activities, i.e., tasks are then marked for other resources (WRP21). Concrete work on tasks can be started afterwards (WRP22). Activities can, however, also be started directly as soon as they have been enabled (WRP23). Work list items can't be sorted by the process model (WRP24), however, by process participants (WRP25). A resource can perform several tasks at the same time (WRP42) and decide herself which task to perform next (WRP26). Tasks can be released after reservation (WRP29) and marked as interrupted (WRP32). Activities can be completed directly after the start, i.e., skipped, if this doesn't violate a constraint (WRP33). Without constraining rules an activity can be repeated arbitrary often (WRP34). Repetitions can be constrained if a certain event, e.g., a milestone, has been reached. Resources can perform prework (WRP35) by defining certain temporal sequences only as recommended instead of obligatory. Activities are created and assigned at the same time (WRP36), however, can't be started directly after an assignment (WRP37). Further activities can be enabled directly after the completion of an activity. Assigned activities together with their current agent can be visualized to all resources (WRP41). An activity is only assigned to a single resource at the same time (WRP43). Summing up, the DPIL framework supports 62% of the organizational and 75% of the data oriented workflow patterns. Therefore, the conducted work shows that the biggest part of workflow data and resource patterns can be implemented by means of DPIL and the DPIL Navigator platform. Note that to the best of our knowledge there is currently no comparative analysis of other declarative language frameworks w.r.t. the Workflow Patterns.

4 RELATED WORK

The work at hand relates to declarative process management with a strong focus on practical applicability of developed concepts. The *Declare* framework was designed for modelling and executing declarative business processes. In its most publicized variant, a Declare process model is built from a set of rule templates each of which is mapped to an expression in Linear Temporal Logic (LTL). The resulting LTL formula is then converted to an automaton for execution (Pesic, 2008). Declare only constrains the starts of activities and interrelates them temporally. Data oriented aspects and the organizational perspective are completely missing in Declare. The approach proposed in (Lamma et al., 2007) allow for the specification of constraints that go beyond the traditional Declare templates. In (Westergaard and Maggi, 2012), the authors define *Timed Declare*, an extension of Declare that relies on timed automata. In (Montali et al., 2013), the authors introduce for the first time a data aware semantics for Declare. In (Burattin et al., 2015) a general multi perspective LTL semantics for Declare (*MP-Declare*) has been presented. Here, Declare is extented with elements of first order logic to refer to data values in constraints. Data aware as well as generalized MP-Declare models are currently only supported in the context of conformance checking (Burattin et al., 2015) and process discovery (Schönig et al., 2016). A system supported modelling and execution is currently not possible. *CLIMB* (Montali, 2010) is a first-order logic declarative language for the specification of interaction models. Here, the Declare is also extented with further process perspectives like data and resources. As for MP-Declare, there is no system support for modelling and execution of CLIMB models. The *DCR Graph* framework (Slaats et al., 2013) and graphical representation is similar to the Declare. The DCR Graph model directly supports execution of the process model based on the notion of markings of the graph. The framework lacks system support for data and resource oriented aspects. The Case Management Model and Notation (*CMMN*)[4] represents recent efforts to standardize declarative business process modelling. CMMN neglects the organizational perspective. The performer can only be selected on the basis of a role and the perspective is completely missing in the graphical representation of CMMN models. System support for data in CMMN is currently is still not available.

5 CONCLUSIONS

In this paper, we introduced the declarative process modelling and execution framework DPIL that covers resource and data modelling. Expressiveness and functionality of the framework have been evaluated by means of the Workflow Data and Resource Patterns. We described concrete model excerpts that implement frequently needed patterns. Summing up, the work at hand can serve as an instruction manual for implementing a declarative process management solution in practical applications. Based on the lessons learned the DPIL framework is currently used in several industry projects to digitally support operative processes. Future research in this context will focus on further improving modelling and execution support for DPIL models, e.g., by a web-based modelling tool. DPIL is currently a textual language that might be difficult to read and understand by end users. Therefore, a graphical language and editor is desired.

REFERENCES

Burattin, A., Maggi, F. M., and Sperduti, A. (2015). Conformance checking based on multi-perspective declarative process models. *preprint arXiv:1503.04957*.

Fahland, D., Lübke, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., and Zugal, S. (2009). Declarative versus imperative process modeling languages: The issue of understandability. In *Enterprise, Business-Process and Information Systems Modeling*, pages 353–366.

Jablonski, S. and Bussler, C. (1996). Workflow management: modeling concepts, architecture and implementation.

Lamma, E., Mello, P., Riguzzi, F., and Storari, S. (2007). Applying inductive logic programming to process mining. In *Inductive Logic Programming*, pages 132–146.

Montali, M. (2010). *Specification and verification of declarative open interaction models: a logic-based approach*, volume 56. Springer Science & Business Media.

Montali, M., Chesani, F., Mello, P., and Maggi, F. M. (2013). Towards data-aware constraints in declare. In *SAC*, pages 1391–1396. ACM.

Pesic, M. (2008). Constraint-based workflow management systems: shifting control to users.

Reichert, M. and Weber, B. (2012). *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer Science & Business Media.

Reijers, H. A., Slaats, T., and Stahl, C. (2013). Declarative modeling–an academic dream or the future for bpm? In *BPM*, pages 307–322. Springer.

Russell, N., Ter Hofstede, A. H., Edmond, D., and van der Aalst, W. M. (2005a). Workflow data patterns: Identification, representation and tool support. In *International Conference on Conceptual Modeling*, pages 353–368. Springer.

Russell, N., van der Aalst, W. M., Ter Hofstede, A. H., and Edmond, D. (2005b). Workflow resource patterns: Identification, representation and tool support. In *CAISE*, pages 216–232.

Schönig, S., Ackermann, L., and Jablonski, S. (2017). DPIL Navigator 2.0: Multi-Perspective Declarative Process Executio. In *BPM Demos*.

Schönig, S., Di Ciccio, C., Maggi, F. M., and Mendling, J. (2016). Discovery of multi-perspective declarative process models. In *ICSOC*, pages 87–103. Springer.

Schönig, S. and Zeising, M. (2015). The DPIL Framework: Tool Support for Agile and Resource-Aware Business Processes. In *BPM Demos*.

Slaats, T., Mukkamala, R. R., Hildebrandt, T., and Marquard, M. (2013). Exformatics declarative case management workflows as dcr graphs. In *BPM*, pages 339–354. Springer.

Vaculín, R., Hull, R., Heath, T., Cochran, C., Nigam, A., and Sukaviriya, P. (2011). Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *EDOC*, pages 151–160.

van Der Aalst, W. M., Ter Hofstede, A. H., Kiepuszewski, B., and Barros, A. P. (2003). Workflow patterns. *Distr. and parallel databases*, 14(1):5–51.

Westergaard, M. and Maggi, F. M. (2012). Looking into the future: Using timed automata to provide a priori advice about timed declarative process models. In *OTM*, pages 250–267. Springer.

[4]http://www.omg.org/spec/CMMN/