A State-based Intention Driven Declarative Process Model

*Pnina Soffer

University of Haifa, Carmel Mountain 31905, Haifa, Israel, spnina@is.haifa.ac.il

Abstract. Declarative process models support process flexibility, whose importance has been widely recognized, particularly for organizations that face frequent changes and variable stimuli from their environment. However, the currently dominant declarative approaches lack expressiveness for addressing the process context (namely, environment effects) and leading its execution towards a goal. The paper proposes a declarative model which addresses activities as well as states, external events, and goals. The model is based on the Generic Process Model (GPM), extended by a notion of activity, which includes a state change aspect and an intentional aspect. The achievement of the intention of an activity may depend on events in the environment and is hence not certain. The paper provides a formalization of the model and describes an execution mechanism. It emphasizes the usefulness of specifying the intentional aspect of activities, by using it as a basis for semantic validation of the model at design time and for a planning module that can guide execution at runtime. These are illustrated by an example from the medical domain.

Keywords: Declarative process model, Goal, Intention, Generic Process Model

Introduction

The importance of flexibility in process aware information systems has been widely acknowledged in the past few years. Flexibility is the ability to make changes in adaptation to a need, while keeping the essence unchanged 0(Regev et. al, 2007). Considering business processes, flexibility is the ability to deal with both foreseen and unforeseen changes, by varying or adapting specific parts of the business process, while retaining the essence of the parts that should not be impacted by the variations (Schonenberg et. al., 2008).

Flexibility is particularly important in organizations that face frequent changes and variable stimuli from their environment. For processes that operate in a relatively stable environment, where unpredictable situations are not frequent, flexibility is not essential, as responses to all predictable situations can be defined. However, in the present business environment, where changes occur frequently and organizations have to cope with a high range of diversity, full predictability is rare.

Facing this reality, approaches have been proposed for enabling flexibility in business processes, as reviewed and classified in (Schonenberg et. al., 2008). These include mechanisms of late binding and modeling, where the actual realization of a specific action is only decided at runtime as implemented in YAWL 0(Aalst and Hofstede, 2005), and changes that can be made at runtime to a running process instance or to all instances of the process, enabled in ADEPT 0(Reichert et. al., 2003).

One of the promising approaches is declarative process models (e.g., Declare 0(Pesic et. al., 2007)), which have received significant attention in recent years.

While "traditional" process models are imperative (e.g., BPMN), explicitly specifying the execution order of activities through control flow constructs, a declarative process model implicitly specifies the execution procedure by means of constraints: any execution that does not violate constraints is possible. Using such model, the user can respond to each situation that arises, executing an activity chosen from all the ones available in compliance to the specified constraints. Currently, the most common approach to declarative process specification (although not the only one existing) is based on Linear Temporal Logic (LTL), which sets constraints on temporal relations among activities 0(Pesic et. al., 2007)0. While allowing a high level of freedom, the approach has the following limitations.

First, while the human decision about which action to take is made based on the state at that specific moment, the existing models do not emphasize states. Rather, the leading concept to be modeled and monitored in the model is an activity, and constraints can be specified on the execution of a single activity or on relationships between activity executions. The process state is monitored, mainly as a trace of the activities that have been executed up to a given moment. Constraints can also relate to values of data as conditions for activity execution. However, there is no fundamental view and monitoring of state for leading process execution and decision making.

Second, to respond to changes and events in the environment, these need to be addressed in the model. Generally speaking, the model should be context aware, where context is the set of inputs a process instance receives from its environment0 (Ploesser et. al., 2009). This is particularly important when bearing in mind that flexibility is required in the first place in processes that face frequent changes in the environment.

Finally, an effective selection of action by the human operator of the process should relate to the desired outcomes to be achieved, namely, to a goal. Currently, goals are usually not an integral part of process definitions.

Some proposals of state-based declarative models have also been made (e.g., Hull et. al., 2011). However, these are either missing the concept of goal (Hull et. al., 2011) or of environment effects 0 et. al., 2008). This paper proposes a declarative process model to overcome the three discussed limitations. To develop a consistent and complete model, we rely on the Generic Process Model (GPM) (Soffer and Wand, 2007), which is an ontology-based theoretical process analysis framework. GPM uses states as a leading element in process representation; it has been used for analyzing the context of processes (Ghattas et. al., 2009)0, and it includes goals as basic building blocks of processes. Since GPM emphasizes states and abstracts from activities in process models, in this paper it is amended to cater for activities as well. We model activities by the state change they cause and by the intention that drives them. We use the intentional element of activities to drive validation at process design and as a basis for action selection at runtime.

In what follows, we start by a motivating example, demonstrating the limitations of a representative LTL-based declarative model. We then present the concepts required for our declarative model, first by informally deriving them from GPM, and then as formal definitions that set the basis for an execution mechanism. Then, the use of our concepts at process design and at runtime is presented and demonstrated through application to the running example. This is followed by a discussion, review of related work, and conclusions, also outlining future research directions.

Motivating Example

This section presents a motivating example of a CT (Computed Tomography) virtual cardiac catheterization process, which will be used throughout the paper as a running example. We use Condec (used in Declare 0(Pesic et. al., 2007)0) as a representative LTL-based specification, since models of this kind are currently dominant in the declarative model research. A Condec model of the process is given in Figure 1. The process starts with a pre CT evaluation of the patient (marked in the model as “init”). This evaluation may find the patient not fit for the scan, so the patient is released and the process ends. If the patient is fit and is not regularly on beta blockers, he will be administered beta blockers as preparation for the scan. Following this, either obesity (for overweight patients) or regular cardiac CT scan is performed once. The scan uses a low level of radiation, serving as a first indication of arteriosclerosis. If an evidence of

calcification is obtained, the patient is released. If the first scanning does not discover an evidence of calcification, a second scanning is performed. Scanning can be performed up to twice (marked in the model as "0..2" for the scan activities), and if the second scan fails, the patient is released. In case the second scan is successful, its results are deciphered and interpreted. Deciphering can be successful or unsuccessful; if it is unsuccessful, there might still be a possibility to get better results, so it can be tried again. When deciphering ends (successfully or not) the patient is released. At any point in the process, some acute health situation might be identified, so the patient is immediately sent to an emergency room (ER) and the process ends. In the model this is represented as ER intervention, with an exclusive choice relation with Release patient (both end the process under different circumstances). The patient might also feel bad during the process (due to allergy, irregular heart rate, etc.). In such cases the procedure may be paused for a while and resumed when the patient feels better. The alternate precedence relation between these two activities denotes that each execution of Resume must be after an occurrence of Pause examination. Additionally, at any point in time, the patient may be released so the process ends, but unsuccessfully.

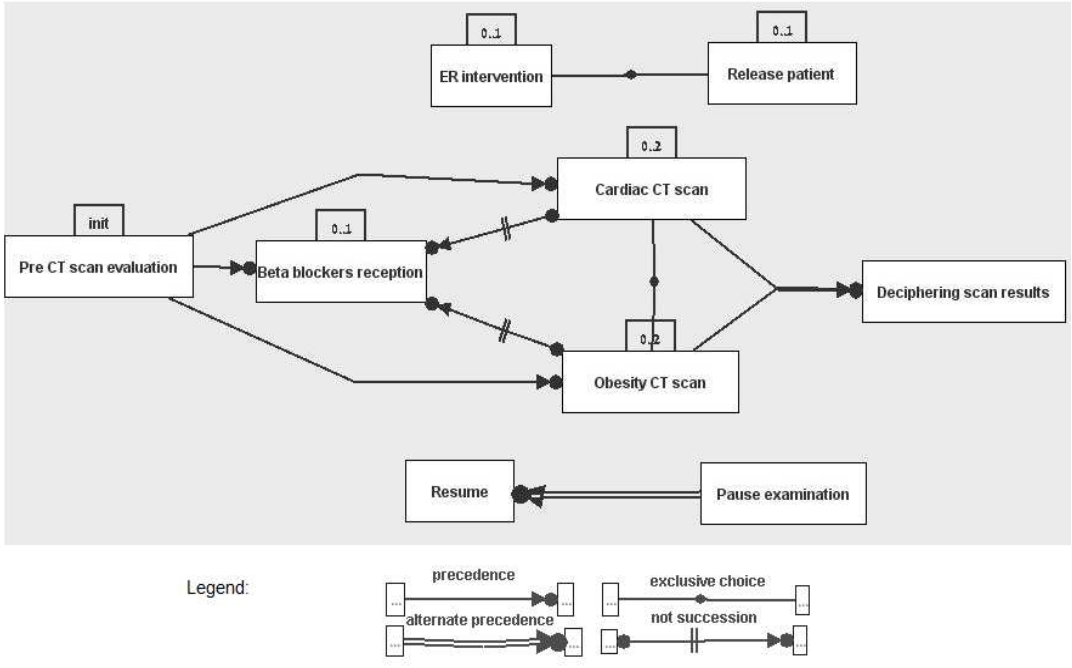

Figure 1: The example process: a Condec model

The Condec model specifies the ordering of the process activities by a precedence relation, denoting that the second activity is only possible after the first one has taken place (but it is not mandatory after the first one). The not-succession relations from Cardiac and Obesity CT scan to Beta blockers reception denote that Beta blockers cannot be administered after the scan (hence if it should be performed, it should be before the scan). The activities of ER intervention, Release patient and Pause examination may (or may not) take place at any point in the process, and are not preceded by other activities. The existence constraints above activities specify the number of times an activity must or can be performed (e.g., 0..1). Activities whose existence is not explicitly constrained, can be performed any number of times (or none).

The Condec representation supports the flexibility required for the process, catering for unforeseen situations and providing an immediate response based on human decision making. Note that the process could also be specified using an imperative modeling language (e.g., BPMN). However, this would require a very complex model to specify all the possible variations.

Despite all the discussed advantages, we claim that this representation is not expressive enough, and it leaves parts of the flow logic to human judgment, although this logic is clear and needs to be specified and enforced. Examples include: (1) a second scan is performed only if a clear evidence of calcification has not been obtained in the first scan; (2) Beta blockers reception is needed only for patients who do not use them regularly; (3) Pause examination and ER intervention are performed when the patient has some irregularity or when an acute problem is identified, respectively. These are constraints on the process flow, which cannot be expressed as relationships between activities or existence constraints on the activities. Furthermore, the model does not specify conditions for process termination. Implicitly, the process cannot end before all the existence constraints on its activities are satisfied. However, in our example the only mandatory activity is Pre CT evaluation, while termination of the process should be under defined conditions.

Roughly speaking, we may conclude that Condec does not support constraints that relate to the context of the process and to its goal, where context refers to all environmental effects on a specific execution of the process. These may be general environment conditions (see 0 Ploesser et. al., 20090) or specific case properties (Ghattas et. al., 2009). In Condec they are assumed to be addressed by human judgment when the process is executed, enabled by the flexibility of the specification. Furthermore, without an explicit goal specification, the achievement of a goal is completely relying on the human operator.

In the following sections we present an approach derived from theory, which enables a process specification that captures contextual constraints and process goals, while supporting flexibility. The theoretical basis provides for a set of constructs that are capable of fully expressing the business logic of processes.

Ontological State-based View

This section introduces a declarative model based on the Generic Process Model (Soffer and Wand, 2005; 2007), which is a process analysis framework, building on Bunge's ontology (Bunge, 1977). GPM emphasizes states, events, and goals, which, as shown above, are not well addressed in current declarative process models.

The focus of attention in GPM is the *domain* where the process takes place. The process domain is a *composite thing*, represented by a set of *state variables*. State variables can be *intrinsic* to things (e.g. height) or *mutual* to several things (e.g. a person works for a company). The values of the domain state variables at a moment in time denote the *state* of the domain. A state can be *unstable*, in which case it will transform according to the *transition law* of the domain (*internal event*), or *stable*, namely, it will not change unless invoked by an event in the environment (*external event*). Events in the environment affect the domain through mutual state variables of the environment and the domain. GPM views an enacted process as a set of state transitions in the process domain. Transitions result either from *transformations* within the domain (reflecting its transition law), or from actions of the environment on the domain. A process ends when the domain reaches a desired (*goal*) state, which is stable and where no more changes can occur due to domain dynamics.

A process model is an abstract representation of the process, defined as follows.

Definition 1 *(GPM process model): A process model in a given domain is a tuple <I, G, L, E>, where*

I: the set of possible initial states – a subset of unstable states of the domain.

G: the goal set – a subset of the stable states reflecting stakeholders' objectives.

L: the transition law defined on the domain – specifies possible state transitions as mappings between sets of states.

E: a set of relevant external events that can or need to occur during the process.

Note that sets of states are usually specified as a partial assignment of values or as conditions that should hold on the values of part of the domain state variables.

As noted, the focus of attention in GPM is the process domain. The domain boundaries are set to distinguish what is fully controlled by the process and its operators, and what is not. This distinction enables defining the context of a process 0as the set of environmental effects on the process, which are twofold. First, the properties of the specific case handled by a process instance – these are assumed to exist at the initiation of the case, although not all their values are necessarily known then. Second, actions of the environment during process execution, manifested as external events. External events are events (state transitions) in the environment of the process domain, which affect the state of the domain through mutual state variables. Taking place outside the domain, they are not controlled by it. The occurrence of an external event can be unanticipated, but even for anticipated occurrences, the exact time and resulting state are usually not predictable. In particular, they differ for different process instances.

As a basic part of the model, GPM explicitly addresses the goal of a process, enabling the design of a process to achieve its goal. At runtime, achieving a goal state marks the completion of a process instance.

However, the transition law of GPM, which is a mapping between sets of states, is an abstract notion. Specifically, as indicated in Definition 1, GPM's process model abstracts from activities, which are how state changes are brought about. Hence, to make GPM an appropriate basis for declarative process models, the law needs to be decomposed into activities and constraints. To do so, a clear understanding of what an activity is needs to be developed.

Activities are the means for achieving internal events. Since internal events usually affect a subset of the domain state variables, namely, a sub-domain, and since different internal events can occur concurrently in independent sub-domains (Soffer et. al., 2010), an activity is an internal event in a sub-domain. However, a sub-domain may change its state through a series of internal events in an almost continuous manner. What makes a specific trajectory be considered an activity is the intention that drives it. For example, the activity of Pre CT evaluation entails actions such as measuring the patient's blood pressure and heart rate, performing an electrocardiogram, and others. We consider all these actions as parts of one activity, distinguished by the one aim to be achieved. Intentions can be of achieving, maintaining, or avoiding a state (Lamsweerde, 2001).

We define an activity as an internal event in a sub-domain, intended to achieve a defined change in its state. According to our model the state change brought by an activity is deterministic, governed

by the law. However, as illustrated in Figure 2, the state variables whose values are changed might be mutual state variables of the process domain and its environment. In such cases, the environment is affected and its state might become unstable. This, in turn, causes transformations (events) in the environment, and these events might, again, affect the process domain. Thus, an activity that acts on the environment might lead to an external event in response. Since external events are not controlled by the process domain and their outcome is unpredictable, this might seem as if the outcome of the activity is unpredictable (especially if the reaction is immediate). Nevertheless, we specify only the controllable change within the process domain as part of the activity, and distinguish the uncontrollable change as an external event invoked by the activity and its effect on the process environment. Recall, external events are part of the process context. Hence, the result of an activity would depend on a combination of intention and context.

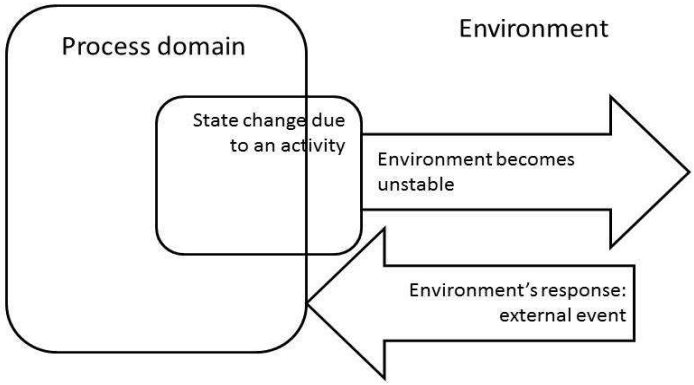

Figure 2: Activity and external event in response

As an example, consider a basketball player throwing the ball to the basket. The activity ends once the ball is in the air, which is an unstable state of the environment. The movement of the ball in the air is not controlled by the player. The resulting event can be that the ball either misses or hits the basket, and it is an external event, not completely predictable. The actual value resulting from the external event will be determined at runtime. Note that in this example, the intention of the activity was to bring about a state where the ball is in the basket, but this can only be achieved by an external event, and with uncertainty.

Activities can hence be classified to two classes: (a) activities that affect only state variables which are intrinsic to the process domain. Such activities cause a fully predictable change that achieves the intention associated with the activity. (b) Activities that affect the state of the environment and

invoke an external event. For these activities the specified (and predictable) change in the state does not necessarily correspond to the intended change. It may not even relate to the same state variables.

We now consider constraints. A GPM process model can be represented by three types of constraints: (a) initiation constraints, setting the possible initial set of states, (b) transformation constraints that specify the relationship state-activity, namely, states that are preconditions for activities, (c) Termination (goal) constraints, defining the set of states where the process can terminate. In addition, we can define a fourth type of constraint – environment response constraints that place external events as response to activities that affect the environment.

Initiation constraints – determine values of state variables to specify the conditions under which the process can begin (e.g., when a patient arrives at the clinic). Note that the initiation constraints do not determine the exact initial state of a process instance. Rather, they are partial assignment of state variable values. The exact initial state also relates to contextual properties which characterize each specific case (e.g., the weight of a patient).

Transformation constraints – include two kinds of constraints: enabling constraints and triggering constraints. Enabling constraints relate activities to the sets of states when they can be activated. Note that the state that follows the execution of an activity is directly calculated from the state that precedes it and the change it causes.

Triggering constraints specify sets of states when an activity must be activated, so when a state in this set is reached the activity will immediately fire.

Termination constraints – determine sets of states where the process terminates. There might be two kinds of termination states. First, goal states, which are stable states the process is intended to achieve. Once a state in the goal set is reached, the process terminates. Second, exception states, which are stable states where the process terminates without achieving what it is intended to achieve. For example, the virtual cardiac catheterization process can terminate when it is found out the patient is not fit for scan so he is released, or when an ER intervention is needed. These are exception states. Note that the process may include stable states which are not defined as termination states. If such a state is reached, the process waits for an external event to reactivate it. In the virtual cardiac catheterization process a state after the examination has been paused is stable, waiting for an external event (patient feels better) to resume the process.

Environment response constraints – relate external events to activities that invoke them. Note that external events can also occur unexpectedly. In many cases the external event does not necessarily

immediately follow the activity; there might be some time elapse between them, but response will eventually occur, and it is part of the process context.

Finally, it can be shown that the combination of initiation, termination, triggering, and enabling constraints is sufficient for expressing all the constraint types available in Condec. Our set of constraints provides these operations with respect to a broader scope, including context and goal. Hence, it provides a richer expressive power.

Formalization

Following the above discussion, we now formalize the proposed constructs.

Definition 2 *(process model): Let D be a domain represented by its state variables vector* $X=(x_1,x_2,\ldots x_n)$*. Let* V_i *be the domain of values of state variable* x_i*,* $V=(V_1,V_2,\ldots V_n)$*. A process model M over D is a tuple (I, G, A, BC, E), where*

I: a set of states satisfying the initiation constraints

G: a set of states satisfying termination constraints; $G=Gg\cup Ge$*; Gg includes states defined as the goal of the process, Ge are states of exceptional termination.*

A: a set of activities

BC: a set of behavior constraints

E: a set of external events.

Sets of states are specified by predicates over the state variable vector. The predicates can require partial assignments of state variable values in X to become true. Hence, given predicates C_I, C_{Gg}, and C_{Ge} that specify initiation, goal, and exceptional termination conditions respectively, we obtain:

$I=\{s|C_I(X)=TRUE\}$; $Gg=\{s|C_{Gg}(X)=TRUE\}$; $Ge=\{s|C_{Ge}(X)=TRUE\}$.

As discussed in the previous section, activities are intentional changes in the state of a sub-domain. Following this, the specification of an activity includes two elements: the change (delta) it brings about to the state of the sub-domain and the intended set of states to be achieved.

Definition 3 *(activitiy): Let* $\delta(X)$ *be a function,* $\delta: V\rightarrow V$*. Then* $a\in A$*:* $(\delta(X),\gamma(X))$*, where* γ *is a predicate denoting the set of states intended to be achieved by the activity.*

Note that δ usually implies a change in a subset of the domain state variables, which are the ones affected by the activity. Also note that if γ(X) includes negation operators, then the intention of the activity is to avoid a set of states. As well, if γ(X) refers back to the set of states that precede the activity, then the intention of the activity is to maintain an existing state.
The set of behavior constraints includes the transformation and environment response constraints. As discussed in the previous section, transformation constraints include enabling constraints and triggering constraints.

Definition 4 *(enabling constraint): Let* $a \in A$, θ_a *a predicate,* $En(a)=\{s|\theta_a(X)=TRUE\}$, *then a can fire for every* $s \in En(a)$.

Definition 5 *(triggering constraint): Let* $a \in A$, τ_a *a predicate,* $Tr(a)=\{s|\tau_a(X)=TRUE\}$, *then a must fire for every* $s \in Tr(a)$.

Note that a triggered activity must also be enabled, hence $Tr(a) \subseteq En(a)$. In order to define the environment response constraints, we first need to define the external events element in the model. In a process model an external event is an occurrence we make no a-priori assumptions about (e.g., regarding its effect on the state of the domain). However, some external events which are expected to occur are expected to affect a subset of the domain state variables and assign them some value within their domain of possible values. The actual state that follows an external event will become known at runtime as input made by the user.

Definition 6 *(external event): An external event* $e \in E$: $\{(x_i,v_i)|x_i \in X, v_i \in V_i\}$

In words, an event is defined by a subset of the domain state variables which it affects, resulting in values within their domain of values.
Environment response constraints relate expected external events to the activity that invokes them.

Definition 7 *(environment response constraint): Let* $a \in A$, $e \in E$. *An environment response constraint Er:(a,e) denotes that e always occurs eventually after a.*

Since the occurrence of external events is not controlled by the process, environment response constraints cannot be enforced at runtime. Nevertheless, they are specified so they can be considered at process design time, and can be taken into account when planning ahead in runtime.

Specifying a process

This section demonstrates how the proposed model can represent the running example. We start by defining the state variables of the domain and their possible range of values (Table 1). Table 1 also provides the initial value of each state variable, defining the initial set of states of the process, *I*. Note that initial values are set for a subset of the state variables, while state variables whose initial value is not specified stand for contextual properties. These need to be initialized to represent specific case properties. In our example process the relevant contextual properties are overweight of the patient and whether the patient is regularly on beta blockers. Also note that the table does not include state variables that count the executions of each activity. All the activity counters are initiated to 0.

Table 1: State variables in the example process and their initial values

State variable	Values	Initial value	State variable	Values	Initial value
Candidacy	{Null, Fit, Not fit}	Null	Beta Blockers	{Given, Not given}	
Over-weight	{Yes, No}		Calcification	{Found, Not found}	Not found
Images	{Null, Successful, Unsuccessful}	Null	Deciphering results	{Null, Successful, Unsuccessful}	Null
Acute problem	{Undiscovered, Discovered}	Undiscovered	Irregularity	{Null, Appear, Disappear}	Null
Patient released	{Yes, No}	No	ER Intervention	{Yes, No}	No
Paused examination	{0, 1}	0			

The termination set is comprised of two sets of states, *Gg* of desired (goal) states and *Ge* of undesired termination states. Considering our example:

Gg={s|(DecipheringResults=“successful”) ∧ (PatientReleased=”Yes”)}

Ge={s|((DecipheringResults≠“successful”)∧(PatientReleased=”Yes”))∨ (ERIntervention=”Yes”)}

Ge stands for two possible cases of termination – when the patient is released without having reached successfully deciphered images (e.g., not found fit to scanning, or after calcification has been discovered), or when ER intervention is needed.

The activities of the process are specified in Table 2 in terms of the function δ, relating to specific state variables, and the predicate γ. The table also specifies for every activity *a* the predicates θ_a and τ_a that define the related transformation constraints *En(a)* and *Tr(a)*, respectively.

Table 2: Activities and corresponding transformation constraints

Activity	**δ**	**γ**	**Transformation constraints**
PreCT Evaluation	PreCTEval.Count → PreCTEvaluation.Count + 1	Candidacy= “fit”	θ_a: (Candidacy=“null”)∧ (Paused Examination=0)
Beta Blockers Reception	(BetaBlockers Reception.Count →Beta blockers Reception.Count +1) ∧ (Beta blockers = "Given")	Beta blockers = "Given"	θ_a:(Compatibility="Fit") ∧ (Beta blockers≠"Given")∧ (Paused Examination = 0)
CardiacCT Scan	CardiacCTScan.Count → CardiacCTScan.Count +1	(Calcification ="Not Found") ∧ (Images = "Successful")	θ_a: (Compatibility="Fit") ∧ (Beta blockers = "Given") ∧ (Over-weight = "No") ∧ (Calcification = "Not Found") ∧ (ObesityCTScan.Count = 0) ∧ (Paused Examination = 0) ∧ (CardiacCTScan .Count < 2)
ObesityCT Scan	ObesityCTScan.Count → ObesityCTScan.Count +1	(Calcification ="Not Found") ∧ (Images = "Successful")	θ_a: (Compatibility="Fit") ∧ (Beta blockers = "Given") ∧ (Over-weight = "Yes") ∧ (Calcification = "Not Found") ∧ (CardiacCTScan.Count = 0) ∧ (Paused Examination = 0) ∧ (ObesityCTScan.Count < 2)
Deciphering Scan Results	DecipheringScanResults. Count → DecipheringScan Results.Count + 1	Deciphering results= “Successful”	θ_a: Images = "Successful"
ER Intervention	ER intervention = ”Yes”	ER intervention = ”Yes”	θ_a: Patient released = “No” τ_a: Acute problem= "Discovered"
Release Patient	Patient Released = “Yes”	Patient Released = “Yes”	θ_a: (ER intervention=”No”) ∧ (Paused Examination = 0)
Pause Examination	Paused Examination → 1	Paused Examination=1	τ_a: ((Irregularity = Appear) ∧ (Paused Examination = 0))
Resume	Paused Examination → 0	Paused Examination=0	τ_a: ((Irregularity = Disappear) ∧ (Paused Examination = 1))

To illustrate the specification of activities and their related transformation constraints, let us consider the activity ObesityCTScan, whose δ relates to the execution counter of the activity, raising it by 1. Recall, this activity can be performed up to twice. Ideally, after two executions a state will be reached where γ is achieved, namely (Calcification="Not Found") ∧ (Images = "Successful"). The enabling set of this activity is when (Compatibility="Fit") ∧ (Beta blockers =

"Given") ∧ (Over-weight="Yes") ∧ (Calcification="Not Found") ∧ (CardiacCTScan.Count=0) ∧ (Paused Examination=0) ∧ (ObesityCTScan.Count< 2), denoting that (a) the activity can start after beta blockers are given (either in the process or in its context) and compatibility is evaluated and found fit (this condition is needed in case beta blockers are given contextually), (b) the activity is executed only for patients with over-weight (in which case CardiacCTScan cannot be performed), (c) the activity can only be performed twice, and it is not repeated if calcification is found, and (d) the activity cannot start when the examination is paused.

Note that there are activities such as BetaBlockersReception, where for a state preceding the activity $s \in En(a)$, the change achieved with certainty $\delta_a(s)$ satisfies the intention γ_a. These are activities that achieve their intention with certainty, not depending on external events. For other activities (e.g., PreCTEvaluation), an external event is expected in response to the activity, for a state satisfying γ_a to be achieved. As previously discussed, in these cases the intention of the activity may not be achieved, depending on the values set by the external event.

To complete the process specification, we define the set of external events *E*, which, together with the uninitiated variables in Table 1, form the context of the process. Table 3 includes external events and their associated environment response constraints (column “Response to” in the table). Some of the external events are expected in response to specific activities, while some can occur unexpectedly, in which case the “Response to” column is blank.

Table 3: External events in the example process

External event	**Affected state variables**	**Response to**
Candidate compatibility	Candidacy	PreCTEvaluation
Calcification discovery	Calcification	CardiacCTScan/ ObesityCTScan
Image generation	Images	CardiacCTScan/ ObesityCTScan
Deciphering outcome	Deciphering results	DecipheringScanResults
Acute health problem	Acute problem	
Irregularity appearance	Irregularity	
Recovery	Irregularity	

To illustrate, consider the event Calcification discovery. This event is expected in response to a CT scan (either obesity or regular). It may change the value of the state variable Calcification from Not Found to Found (see Table 1).

Execution engine mechanism

The execution of the model is based on an Event-Condition-Action (ECA)-like logic, which responds to events and controls the activities that are enabled or triggered at each moment. The ECA rules depict the process constraints so for a given state *s*, the set of enabled activities is $A_{En}=\{a|s \in En(a)\}$, and the set of triggered activities is $A_{Tr}=\{a|s \in Tr(a)\}$. Still, activities are started and completed by users (humans), who decide which activity to execute from the set of enabled or triggered activities, as reflected in the status of the activities.

The possible statuses of an activity are: Disabled, Enabled, Triggered, Active, Paused. Their possible transitions are depicted in Figure 3.

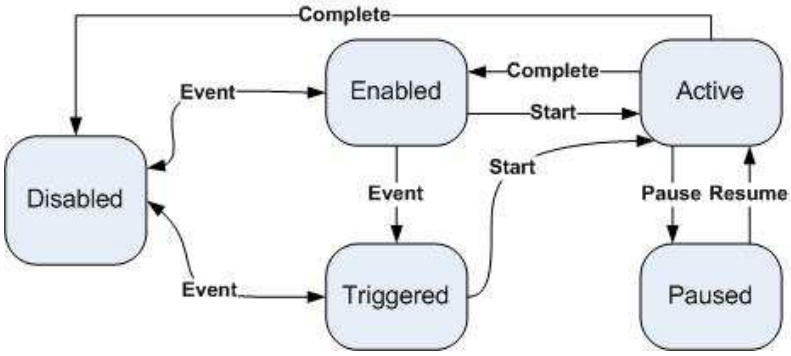

Figure 3: Activity status types and possible transitions

At the beginning of the process, an activity can be enabled or disabled (depending on its transformation constraints). When the state changes through events, disabled activities can become enabled or triggered (as noted, a triggered activity is also enabled), and enabled activities can become triggered as well. It is also possible for an enabled or triggered activity to become disabled (e.g., if another activity, with which the current activity has an exclusive choice relation, was executed). These transitions are governed by the ECA logic. When a user decides to start an enabled or triggered activity, the activity becomes active. While it is active, it can be paused and resumed by the user. When the user completes the activity, the ECA logic sets its status to disabled or to enabled again, based on the transformation constraints. In addition, on activity completion, the new state of the domain is determined, so the state on completion of *a* is $\delta_a(s)$. Finally, the process is explicitly started and terminated by the user. Termination is only possible in a termination state, namely, a state in G. When a process instance is started, the state variable values

are initiated according to the initiation constraints, and an input form is opened for the user to report values of contextual variables.

The following event types are possible: Start(activity), Complete (Activity), Pause (activity), Resume (activity), Event(EventName). The events of start, complete, pause, and resume use the activity name as a parameter; Start and Complete can also relate to the entire process. The general type of event relates to external events, which can assign values to state variables (given as input by the user). For convenience, it is possible to create a set of predefined event templates (EventName), for which it is known in advance which state variables will be assigned a value (e.g., scan results should relate to the state variables of Calcification and of Images).

Since the occurrence of external events should be reported to the system, an event is represented as an input form, where the user can assign values to state variables. The input form is opened when the user decides to report an event. Then he can select reporting a predefined event (EventName) or a general external event, in which case the input form allows assigning values to any state variable. When the user submits the event form, the system can react to the event.

The ECA rules include generic rules (or rule templates), that can be specialized for the specific process and its activities at design time. Below we present these templates, including activity rules, state assessment rules, and termination rules.

Activity rules include rules for the start, complete, pause, and resume events of the activity.

```
On Start(a) if (a.status==enabled OR a.status==triggered) do a.status=active
On Complete(a) do s=δa(s) AND a.status=disabled
On Pause(a) if a.status==active do a.status=paused
On Resume(a) if a.status==paused do a.status=active
```

The rules of state assessment fire at any kind of event (including Start and Complete of activities) and determine which activities should be enabled, triggered, or disabled at the new state that follows the event. Note that the first event (Start (process)) is automatically generated at initiation, once the user submits the form where context variable values are entered.

Event() if (θ_α AND a.status==disabled) do a.status=enabled

Event() if (τ_α AND (a.status==disabled OR a.status==enabled)) do a.status=triggered

Event() if ($\neg\theta_\alpha$ AND a.status==enabled) do a.status=disabled

Event() if ($\neg\tau_\alpha$ AND a.status==triggered) do a.status=disabled

Finally, termination rules include one rule which should fire on an explicit termination of the process. This should be possible only if the current state is in G.

`Complete(process) if (`C_{Gg} `OR` C_{Ge}`) do Terminate(instance)`

Termination of a process instance includes disabling all activities, removing the process instance from the work list and archiving its information.

The role of activity intentions along process life-cycle

The execution engine described in the previous section does not use the intentional part of the activity specification at all. Relying only on state-related constraints, the execution engine's operation is relatively simple. Still, the activity intention information can be utilized at process design time as well as at run time. This section presents these two uses. Below, we first elaborate how intentions can support the validation of a process model at design time. Second, we indicate a possible use of the intention information for planning at run time.

Design time support

At design time, there is a need to assess the validity of a process specification, namely, the ability of the specified process to achieve its goal. We now present five conditions, which are necessary for the specification to be valid, and indicate how to evaluate their satisfaction.

Condition 1 ***(concurrency):*** *For every a, a'∈A, if* $Tr(a) \cap Tr(a') \neq \varnothing$ *then* δ_a *and* $\delta_{a'}$ *do not affect the same state variables.*

This condition is intended to ensure that two activities that are triggered by the same state (namely, must be performed concurrently), do not change the values of shared state variables. For a discussion of this condition, see Soffer et. al. (2010). Verifying the condition is rather straightforward: (a) identifying the subset of activities that have triggering constraints, (b) pair-wise comparison of the triggering conditions of the activities in the set, (c) for couples with overlapping triggering states – check the state variables included in the δ specification. Our example process includes three activities that have triggering conditions: PauseExamination,

ResumeExamination, and ERIntervention. Their triggering constraints are not overlapping, hence Condition 1 is not breeched.

Condition 2 ***(sequence of intended states):*** *There exists at least one sequence of states* $(s_1, s_2,\ldots s_n)$ *such that* $s_1 \in I$, $s_n \in Gg$, *and for* $i=1\ldots n\text{-}1$ $s_{i+1} \in \{s|\gamma_{ai}=TRUE\}$ *and* $s_i \in En(a_i)$.

This condition requires the existence of at least one sequence of states leading from an initial state to the goal of the process, where the enabling or triggering set of each activity is in the intended set of the previous one. This means that the activity is enabled either immediately and certainly after an activity whose intention is achieved by its δ, or after an external event responding to the previous activity has achieved its intention.

Note that there might be other sequences which do not lead to the goal set. These, however, should end on a termination state in Ge and not on any other state. In other words, continuation of the process should be enabled for any state which is not in G. To ensure this continuation, we require the following three conditions.

Condition 3 ***(process continuation – intentions):*** *For every intention* γ *that can be achieved, there exists at least one sequence of states* $(s_1, s_2,\ldots s_n)$ *such that* $s_1 \in \{s|\gamma=TRUE\}$, $s_n \in G$, *and for* $i=1..n\text{-}1$ $s_{i+1} \in \{s|\gamma_{ai}=TRUE\}$ *and* $s_i \in En(a_i)$.

Condition 3 is intended to ensure that for every intention achieved there should be a sequence of states leading to a termination state. The intention is not necessarily on the sequence identified as satisfying Condition 2. Additionally, the requirement is not for the sequence to lead to a goal state, since this might not always be feasible. Rather, the sequence is required to lead to a termination state, ensuring that the process does not get "stuck".

Condition 4 ***(process continuation - activities):*** *Let* $a \in A$ *such that* γ_a *is not achieved by* δ_a. *Then* $\exists$ *an external event* $e \in E$ *and an environment response constraint* $Er:(a,e)$.

Condition 4 is intended to ensure the possibility of every activity to achieve its intention, even if it is not with certainty.

We now introduce a possible way for evaluating Conditions 2, 3, and 4, using a simple Intention-action diagram. Figure 4, presenting an Intention-action diagram of the running example, is also used for informally introducing the semantics of such diagrams.

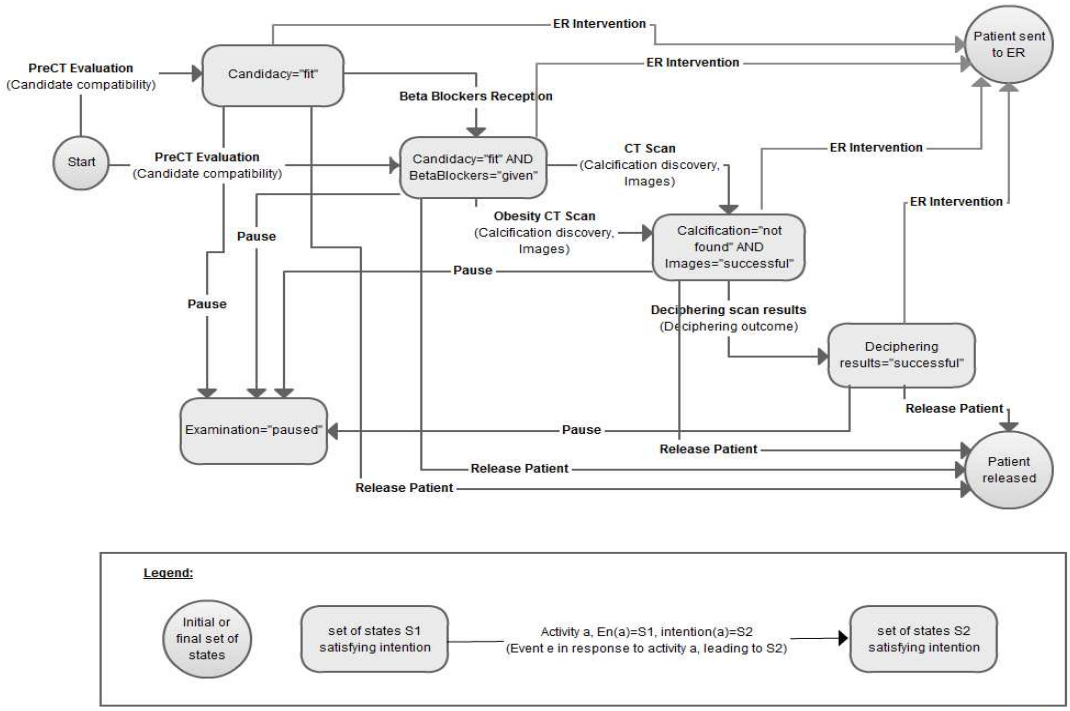

Figure 4: Intention-action diagram of the running example

The notation of the diagram is given in the legend. Note that an arc between sets of states S_1 and S_2, denoting an activity enabled at S_1 and intended to reach S_2, is marked by the activity name; in case the intention is not achieved directly by the activity, but can be achieved by an external event which is a response to that activity, the event (or events) name appears in brackets below the activity name. Also note that the diagram does not differentiate enabling and triggering states. It includes only activities that are enabled at intended states, and does not include activities that are only enabled as a result of external events (e.g., Resume).

Using the diagram, the evaluation of Conditions 2, 3, and 4 is as follows:

Condition 2 – a path from the initial set of states to the goal can be visually detected. Such path can, e.g., take PreCTEvaluation to achieve Candidacy=”fit” AND Beta Blockers=”given”, then through CardiacCTScan reach Calcification=”not found” AND Images=”successful”, DecipheringScanResults to reach Deciphering Results=”successful”, and ReleasePatient, leading

to a state in Gg. Note that this is not the only possible path, but one is sufficient for a positive evaluation of the condition.

Condition 3 – similarly to Condition 2, paths should be established from every intention to a final set of states. In Figure 4 it is possible to see that the condition is evaluated negatively, since no such path exists from the intention Examination="paused". This intention has only incoming arcs and no outgoing ones, since once the examination is paused nothing can be done until an external event of Recovery, which triggers the activity of Resume. However, this external event is not certain to occur. Hence, for the process to always complete, the model needs to be modified, so at least one activity is enabled even when the examination is paused. It would make sense, for example, to change the enabling set of ReleasePatient so it can be performed even when the examination is paused (in case Recovery does not occur after a while).

Condition 4 – directly available in the diagram, through the response external events in brackets next to the relevant activities (e.g., Deciphering outcome next to DecipheringScanResults).

Finally, Condition 5 is also intended to ensure process continuation, but considering the external events rather than intentions and activities. Hence, it needs to be evaluated separately.

Condition 5 ***(process continuation – events):*** *For every external event e: $\{(x_i, v_i)\}$, for every value v_i that state variable x_i can assume, the resulting state s satisfies (1) $s \in G$ or (2) $\exists$ activity $a \in A$ such that $s \in En(a)$.*

Table 4. Enabled activities following external events

External event	**Possible state variable values**	**Enabled or Triggered activities**
Candidate compatibility	Candidacy=Fit	a≠PreCTEvaluation
	Candidacy=Not Fit	ReleasePatient, ERIntervention, PauseExamination, Resume
Calcification discovery	Calcification=Not Found	All activities
	Calcification=Found	ReleasePatient, ERIntervention, PauseExamination, Resume
Image generation	Images=Successful	DecipheringScanResults, ReleasePatient, ERIntervention, PauseExamination, Resume
	Images=Unsuccessful	ReleasePatient, ERIntervention, PauseExamination, Resume
Deciphering outcome	Deciphering results= Successful, Unsuccessful	ReleasePatient, ERIntervention, PauseExamination, Resume
Acute health problem	Acute problem=Discovered	ERIntervention
Irregularity appearance	Irregularity=Appear	PauseExamination
Recovery	Irregularity=Disappear	Resume

This condition requires that every external event either leads to a state in the termination set or to a state where at least one activity is enabled. To evaluate it, all possible state variable values that might be assumed as a result of external events need to be examined. Table 4 indicates the enabled or triggered activities for every external event result. Note that the activities listed in Column 3 of the table are the ones for which the relevant value is part of the enabling set definition. This does not mean that they become enabled when this value is assumed. Rather, they are enabled when this value is in combination with the rest of their enabling conditions. Since Table 4 lists enabled activities for every row, Condition 5 is positively evaluated.

Planning at run time

In addition to the basic execution mechanism described in the previous section, a separate planning mechanism can support the user in deciding which activity to perform at a given moment. The full details of this mechanism are outside the scope of this paper. This section briefly outlines the underlying principles of such planning, and highlights the role of activity intentions for this purpose.

At run time, there might be several activities enabled at any given moment. Current models let the human user make the decision of which activity to perform, based on his judgment and considerations. Implicitly, it is assumed that the human user is goal seeking, hence will select the activity that is most likely to contribute to the goal achievement of the process (even if the goal is not explicitly specified). Some approaches for recommending the next action have been proposed, relying on past experience (Aalst et. al., 2010; Schonenberg et. al., 2008) or on simulation (Rozinat et. al., 2009). As opposed to these approaches, planning looks ahead at the possible state transitions and seeks a sequence of transitions that reaches the goal. Additionally, some objective function can be set, so planning looks for such sequence that optimizes the value of the objective function subject to the given constraints. The objective function is generally referred to as cost to be minimized, but it can relate to other factors, such as time, resources, or number of steps to be taken. In our running example no specific objective function exists, so the objective can be to minimize the number of activities that need to be executed for achieving the goal.

There exist a number of planning formalisms (e.g., STRIPS (Bylander, 1994) 0). Here we refer to the SAS+ formalism (Backstrom and Nebel, 1995), where a general planning problem is of the following form:

A SAS+ *planning task* is a 4-tuple $\Pi = \langle V, O, s_0, s^* \rangle$ with the following components:

$V = \{v_1, \ldots v_n\}$ is a set of *state variables*, each with an associated finite domain D_v.

A *partial variable assignment* over V is a function s on some subset of V such that $s(v) \in D_v$ wherever s(v) is defined. If s(v) is defined for all $v \in V$, s is called a *state*.

O is a set of *operators*, where an operator is a pair <pre, eff> of partial variable assignments called *preconditions* and *effects*, respectively. s_0 is a state called the *initial state*, and s^* is a partial variable assignment called the *goal*.

An operator <pre, eff> is applicable in a state s iff s(v) = pre(v) whenever pre(v) is defined. Applying the operator changes the value of v to eff(v) if eff(v) is defined.

It is easily seen that a general planning problem specification is quite similar to our declarative model. The main difference is that a planning problem does not consider external events. Hence, we propose to use planning under the assumption that every activity achieves its intention. With this assumption, it is possible to get an initial plan.

The planning problem would be $\Pi = \langle V, O, s_0, s^* \rangle$, where V is the set of state variables considered, the operators in O correspond to the activities, where pre= En(a) and eff=γ(a) for every activity; s_0=I; $s^*=G_g$.

Two notes should be made. First, the state variables are required to have a finite domain of values. This is reasonable in most practical situations, as even state variables whose domain of values is infinite (e.g., weight) can in most cases be represented on a finite scale of meaningful ranges (e.g., normal weight, overweight). Second, the planning aims at achieving a state in G_g and not just any termination state, to obtain a plan that achieves the process goal.

As noted, the plan would assume every activity to achieve its intention. However, this would not necessarily be what actually happens. When the process is executed, in case the intention of an activity is not achieved (due to the state brought by an external event), re-planning can be performed, taking the current state as an initial state and again seeking a plan for achieving the goal from that point. If no feasible solution is found by the planning algorithm, some other termination state (not in the goal) can be sought.

Discussion

Declarative process models are a promising means for achieving flexibility in business processes. The paper has indicated gaps in the existing declarative models, and proposed a model which includes aspects that are not addressed by other approaches. Below we discuss the advantages gained by this approach, as well as its limitations, as compared to LTL-based declarative models.

Flexibility vs. decision making support: Enabling and triggering activities based on state and context information, and addressing the process goal, the proposed model supports decision making while providing sufficient flexibility when needed. Yet, the flexibility allowed by LTL-based specifications is higher. The Condec specification of the running example allows releasing the patient at any point in the process, or executing DecipheringResults repeatedly again and again. These, naturally, would be avoided by a goal-seeking process operator, but decisions are completely delegated to human and not supported (or enforced) by the model. The absence of a process goal in LTL-based models makes an abundance of execution traces possible, and it is the sole responsibility of the human to lead it towards successful completion.

Activity representation: The model proposed in this paper is novel in modeling the activities by the definite change they bring about and the intention they serve, and separating the non-deterministic effects of external events from the activities that invoke them. Not relating to states, LTL-based models do not address the outcome of an activity (as discussed, this is left for human consideration). Our representation relies on the ontological view, aiming to anchor the process model in real world business semantics and gaining an in-depth understanding of the process and what drives it.

Intention-based life-cycle support: At process design, the intentions can provide a means for semantically validating the process model. LTL-based process models only employ verification techniques (Pesic et. al. 2007), capable of checking the absence of dead activities and conflicting constraints. Semantic validation of existing models is not supported, especially since the business semantics of goal is not depicted in the model. At runtime, intentions can serve for planning and providing recommendations on which activities to execute at a given moment. Recommendation is important when several possible paths can be taken. Planning can also relate to some objective function (e.g., cost or time) and recommend a path that optimizes it. The proposed execution engine, however, ignores the intention element of the activities. Rather, it only relates to the state,

events, and constraints. This separation is intended to simplify and avoid over complication of the execution engine.

State representation dimensionality: As compared to LTL-based models, the main limitation of the proposed model is the high dimensionality caused by addressing the various state variables in the process domain. This applies to both the execution engine and the planning module. Considering execution, the feasibility and applicability of ECA engines has been long demonstrated (e.g., McCarthy and Dayal, 1989). In our case, experimentation is still needed to assess the effect of process size on the performance. Considering planning, finding an optimal solution is a Pspace-complete problem (Backstrom and Nebel, 1995), thus large models might be impossible to optimize at acceptable time. However, in many cases (as in the running example of this paper) optimization is not sought. Rather, the main issue is to find a feasible path to the goal. In other cases, efficient heuristics are available for finding a sub-optimal yet good solution.

Usability: Unlike LTL-based models, the proposed model does not employ a graphical notation. We intend to consider different visualization possibilities, including an adaptation of the graphical notation of Condec used by Declare, to the additional expressiveness of our model. Note that the lack of a graphical notation is mainly an obstacle at process design, while runtime usability can be achieved without an explicit graphical representation of the model. An appropriate visualization is hence expected to increase the usability of the model at process design, and to a lesser extent also at runtime.

Related Work

Substantial research efforts have been invested in declarative process models in recent years. Most notably, Declare 0, an LTL-based modeling and execution platform, which also allows changing the model at runtime and performing some verification. Methodological issues that concern declarative process models have also been investigated. Examples include life-cycle support (Zugal et. al., 2011), user assistance 0 et. al., 2008), and usability evaluation (Weber et. al., 2010). User assistance includes recommendations which are generated based on similar past process executions by considering specific business objectives 0 et. al., 2008). An initial usability evaluation using the Alaska simulator, has indicated that humans are capable of coping with flexibility and can effectively plan in an agile manner 0 et. al., 2010).

Declarative model segments have also been used for managing imperative models. Ly et. al. (2008) developed a framework for integrating constraints into adaptive process management systems in order to ensure semantic correctness of running processes at any time. Awad et al. (2008) suggested a technique to accomplish the verification of process models against imposed compliance rules by using BPMN-Q queries.

Planning based on declarative specification has also been proposed. Barba et. al. (2012) used planning together with a Condec specification, both at build time for obtaining an initial plan and at runtime for re-planning. However, the LTL-based specification does not employ a process goal, so planning aims at achieving a state where all constraints are satisfied, with the objective of minimizing execution time.

All these approaches basically employ an activity-based view, with LTL-based constraints on activities. Goals are usually not specified or addressed, although some goal consideration is included in the Alaska simulator (Weber et. al., 2009). It relates to a goal, which is actually an objective function, as an overall business value, i.e., minimization of cost, cycle time or the optimization of quality or customer satisfaction. Goals of this kind are called soft-goals in GPM terminology, and can be addressed as part of the planning.

A different and early approach is presented by Andersson et. al., (2005) who defined a business process pattern based on the state-oriented approach that includes a state space, a goal, and valid movements in the state space. Constraints are not made formally and explicitly, but roughly in the form of valid movements. Other, more recent approaches, which consider states as well as activities include the declarative model proposed by Rychkova et. al. (2008) and the Guard-Stage-Milestone (GSM) model (0 et. al., 2011) proposed for business artifact lifecycles. The first one proposed using a declarative specification as a first step at process design. The specification is therefore not executable, but it can be refined to an imperative model when the process is customized and deployed. The declarative specification employs a state space with activity pre- and post-conditions. However, it assumes an activity always achieves a predefined state and does not take external events into account. GSM is a declarative specification consisting of nested process stages that have guards and milestones. The guards monitor the conditions for activating the stages, while milestones represent their possible completion states. GSM is capable of addressing external events and considering the state that follows them. It employs an ECA-like execution mechanism. In all these elements, GSM bears much similarity to the model proposed in

this paper. However, GSM does not support an intentional view and its process lacks a clear goal, which would indicate which of the possible milestones is preferable and should be aspired for.

In summary, the approach presented in this paper extends the capabilities of both LTL-based and state-based declarative process models. Compared to LTL-based models, the approach has an extended expressiveness, enabling all LTL-based constraints and adding state and goal information. Considering state-based declarative models, the intentional aspect of both the process (goal specification) and activities enables state-based execution as well as goal-oriented design and planning.

Conclusion

Declarative process models support flexibility in process aware information systems. The paper proposes a model which extends currently available declarative process models. The model constrains the execution of activities based on state, which reflects activity execution as well as case properties and results of events in the environment of the process. As such, it is context aware and suitable for highly diverse and frequently changing environments, where process flexibility is particularly important. Furthermore, an explicit goal specification can guide execution towards this goal and serve for validating the process at design time and planning at runtime.

We consider the main contribution of the paper to be the activity definition, which makes a clear distinction between the certain change brought about by it and the intended change, which may or may not depend on environment response invoked by the activity. The design time validation and planning mechanism demonstrate the usefulness of the intentional information of the activities. Describing the execution mechanism, we show that when unnecessary, the intentional information can be ignored for simplicity, as it has no role in the actual execution mechanism.

We envision further use of the intentional information along the process life-cycle as future research directions. Mining can possibly indicate the success rate in intention achievement over time, so improvements can be made for increasing this rate. Additionally, constraints and relevant state variables can change during the execution of the process. This possibility is supported by Declare, and we consider it as an important future research direction.

References

van der Aalst W.M.P. & ter Hofstede A.H.M. (2005) YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275.

van der Aalst W.M.P., Pesic M., & Song M.S.. (2010) Beyond process mining: from the past to present and future. *In Proc. of CAiSE, volume 6051 of LNCS*, (pp. 38-52). Springer-Verlag, Berlin.

Andersson B., Bider I., Johannesson P. & Perjons E. (2005) Towards a formal definition of goal-oriented business process patterns, *Business Process Management Journal*, *11*(6), 650-662.

Awad, A., Decker, G., & Weske, M. (2008) Efficient compliance checking using BPMN-Q and temporal logic, In: Dumas et. al. (eds), *BPM '08*, *LNCS 5240*, (pp. 326-341). Springer-Verlag, Berlin.

Backstrom, C., & Nebel, B. (1995). Complexity results for SAS^+ planning. *Computational Intelligence 11*(4), 625-655.

Barba I., Weber B. & Del Valle C. (2012) Supporting the optimized execution of business processes through recommendations, *BPM Workshops*, *LNBIP 99*, Part 2 (pp. 135-140). Springer-Verlag, Berlin.

Bunge, M. (1977). *Treatise on Basic Philosophy: Vol. 3, Ontology I: The Furniture of the World*, Boston: Reidel.

Bylander, T. (1994) The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, *69*(1-2), 165-204.

Ghattas J, Soffer P, Peleg M. (2009). A formal model for process context learning. In: *Proc. BPI 2009*, Ulm, Germany.

Hull R., Damaggio E, De Masellis R., Fournier F., Gupta M., Heath III F., Hobson S., Linehan M., Maradugu S., Nigam A., Sukaviriya P., & Vaculín R., (2011) Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events, *Intl. Conf. on Distributed Event-Based Systems (DEBS)*, 2011

Lamsweerde A. (2001) Goal-oriented requirements engineering: a guided tour, *5th Int'l Symp. on RE*, 249-261, IEEE CS Press.

Ly, L.T., Rinderle, S., & Dadam, P. (2008) Integration and verification of semantic constraints in adaptive process management systems. *Data and Knowledge Engineering 64*, 3-23

McCarthy D. R. & Dayal U. (1989) The architecture of an active database management system, In Proc. *ACM SIGMOD Intl. Conf. on Mgmnt of Data (SIGMOD)*, (pp. 215-224). ACM Press.

Pesic M, Schonenberg MH, Sidorova N, & van der Aalst WMP. (2007) Constraint-based workflow models: change made easy. In: Curbera et. al. (eds), *Proc. of OTM*, LNCS 4803, (pp.77–94). Springer-Verlag, Berlin

Ploesser K., Peleg M., Soffer P., Rosemann M., & Recker J., 2009, Learning from context to improve business processes, *BPTrends*, 6(1):1-7

Regev G, Bider I., & Wegmann A. (2007) Defining business process flexibility with the help of invariants. *Software Process Improvement and Practice, 12*. 65–79.

Reichert M., Rinderle S., & Dadam P., (2003) Adept workflow management system. In: van der Aalst et. al (eds), *BPM 2003, LNCS 2678,* (pp. 370–379). Springer-Verlag Berlin. A.

Rozinat, M.T. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, and C. Fidge. Workflow Simulation for Operational Decision Support. Data and Knowledge Engineering, 68(9):834-850, 2009.

Rychkova I., Regev G., & Wegmann A., (2008) Using declarative specifications in business process design, *International Journal of Computer Science and Applications, 5*: 3b, 45-68.

Schonenberg H., Mans R., Russell N., Mulyar N. & van der Aalst WMP. (2008) Process flexibility: a survey of contemporary approaches, In Dietz et. al. (eds), *CIAO! And EOMAS 2008, LNBIP 10*, (pp. 16-30). Springer, Berlin.

Schonenberg H, Weber B, van Dongen BF, & van der Aalst WMP. (2008) Supporting flexible processes through recommendations based on history. In: Dumas et. al. (eds), *BPM 2008, LNCS 5240*, (pp. 51–66). Springer-Verlag, Berlin.

Soffer, P., Kaner, M., & Wand, Y. (2010) Assigning ontology-based semantics to workflow nets", *Journal of Database Management*, *21*:3, 1-35.

Soffer, P. & Wand, Y. (2005) On the notion of soft goals in business process modeling, *Business Process Management Journal*, *11*:6, 663-679.

Soffer, P., and Wand, Y. (2007) Goal-driven multi-process analysis, *Journal of the Association of Information Systems* (8:3), 175-203.

Weber, B., Pinggera, J., Zugal, S., & Wild, W. (2010) Handling events during business process execution: An empirical test. In: *ER-POIS at CAISE*. 19-30.

Weber, B., Reijers, H., Zugal, S., & Wild, W., (2009) The declarative approach to business process execution: An empirical test, In: van Eck et. al. (eds), *CAiSE 2009*, *LNCS 5565*, (pp. 470-485) Springer-Verlag Berlin Heidelberg.

Zugal S., Pinggera J., & Weber B. (2011) Toward enhanced life-cycle support for declarative processes, *Journal of Software Maintenance and Evolution: Research and Practice*.