

# The Resultmaker Online Consultant: Flexible Declarative Workflow Management in Practice\*

Mukkamala Raghava Rao<sup>1,2</sup>, Thomas Hildebrandt<sup>2</sup>, Kaare Nørgaard<sup>1</sup>, and  
Janus Boris Tøth<sup>1</sup>

<sup>1</sup> Resultmaker A/S, Vester Farimagsgade 3, 1606 Copenhagen, Denmark  
{rao, kn, jbt}@resultmaker.com

<sup>2</sup> IT University of Copenhagen, Rued Langgaardsvej 7, 2300 Copenhagen, Denmark  
{rao, hilde}@itu.dk

**Abstract.** We present the patented process model employed in the Resultmaker, Online Consultant (OC) workflow management system as an example of a flexible declarative workflow process model used in practice. We describe and formalize the key primitives of the OC process model (sequential predecessor, logical predecessor, activity conditions and dependency expressions) as Linear time Temporal Logic (LTL) formulas. This supports a recently proposed approach by van der Aalst and Pesic to use LTL as the foundation for flexible declarative process languages and suggests interesting new constraint templates for the ConDec and DecSerFlow languages based on this approach, that take account for dynamic changes in activity conditions and dependency expressions.

## 1 Introduction

Research in the flexibility of workflow management systems deals with the problems of how to maintain freedom to select as many different flows as possible in process specifications, how to accommodate dynamic changes of workflow processes, and the ability to easily reuse process descriptions in different contexts [1, 7].

As pointed out in [8] most work on flexibility of workflow has so far focussed on *imperative* process languages as employed in the majority of the currently used business process and workflow management systems. However, the authors argue that the use of imperative process languages often leads to *over specification*, which imposes too many constraints on the flows and consequently amplify the need for changes to the specified process. Based on this, the authors propose a paradigm shift replacing the imperative process languages with *declarative* process languages, in which one specify the constraints between work activities rather than exactly how these constraints are resolved. Concretely, they propose

---

\* This work was funded in part by the Danish Research Agency (grant no.: 2106-07-0019, no.: 274-06-0415) and the IT University of Copenhagen (the TrustCare and CosmoBiz projects).

the open languages ConDec [8] and DecSerFlow [9], which are based on the idea of using templates for Linear time Temporal Logic (LTL) formulas [5, 6] as the foundation for flexible declarative process languages.

In the present paper we describe the Process Matrix, which is the patented declarative process model employed in the Resultmaker, Online Consultant workflow management system. The work is part of the industrial PhD scholarship of the first author (funded by Resultmaker A/S) and the research projects on Computer Supported Mobile Adaptive Business Processes (CosmoBiz.org) [2] and Trustworthy Pervasive Healthcare Services (TrustCare.eu) [3].

The Process Matrix has evolved from Resultmaker’s industrial experiences obtained during the process of authoring solutions for the Danish public sector, and has been used with success in practice for several years. It is based on a shared data architecture and electronic forms (updating the shared data) as the key basic activity. Hereto comes activities for connecting to external systems, inviting participants and digitally signing data, that we will ignore in the present paper.

The key primitives of the Online Consultant Process Matrix is that of *sequential* and *logical* predecessor relations between activities A and B, and the introduction of *activity conditions* and *dependency expressions* for each activity. That A is a sequential or logical predecessor of B informally means that activity A must be carried out before B can be carried out – and in the case of a logical predecessor, that B must be redone if A is redone after the execution of B. Activity conditions and dependency expressions refer to values of variables in the shared data store and are dynamically evaluated after each step of the workflow. An activity condition determines if an activity is currently included in the workflow (i.e. it is active) and a change in a dependency expression determines that an activity must be re-executed (i.e. due to changes in data on which it depends). Activity conditions make it very easy to reuse a process description for a different purpose in a different variant: One just adds a new boolean variable to the shared data store and use it to toggle the inclusion or exclusion of activities.

It turns out that the Process Matrix can be naturally formalized as LTL formulas. The formulas suggest new interesting variants of the constraint templates given for the ConDec and DecSerFlow languages, which take account for the effect of dynamic changes in activity and dependency conditions. On the other hand, the formalization also suggests extensions to the Online Consultant, in particular one may follow the approach in DecSerFlow and consider allowing designers to specify new process primitives as LTL formulas based on an open set of templates. We thus believe that the study in this paper forms the starting point for a valuable cross-fertilization between development of workflow management systems in practice and research in theoretical computer science.

The structure of the paper is as follows. In Sec. 2 we introduce the Online Consultant workflow architecture and key components, in particular the *Process Matrix* process model. We illustrate by a real case story in Sec. 3 that the primitives of the Process Matrix gives rise to a high degree of flexibility. In Sec. 4 we formalize the key primitives of the OC process matrix as LTL formulas

and relate the resulting templates to the DecSerFlow language. We end in Sec. 5 by a conclusion and outlining future work.

## 2 The Online Consultant

In this section we introduce the Online Consultant workflow architecture and key components, in particular the declarative primitives of the Online Consultant process model, referred to as the *Process Matrix*.

### 2.1 The Online Consultant Architecture

The Resultmaker Online Consultant (OC) is a user-centric workflow management system based on a shared data store and so-called *eForms* as its principal activities. An eForm is a questionnaire presented by the front end Form engine for the human user as a form in a web browser. The fields in the form is mapped to variables in the shared data store. The data provided by the user is stored in the variables in the shared data store after completion of the form, and will be available for all other activities in the process. In addition to eForms it is possible to specify activities which connect via a Script engine to external systems, e.g. for carrying out automated tasks.

Fig. 1 shows the overall architecture of the Online Consultant. What we just described constitute the Run-time Services of the Online Consultant. The Design-time services consist of tools for designing eForms, processes, etc.

### 2.2 The Process Matrix

The Online Consultant workflow engine is based on a patented workflow model referred to as the *Process Matrix*. The Process Matrix model has evolved from Resultmaker's industrial experiences obtained during the process of authoring solutions for the Danish public sector. Below we describe its key elements.

**Activities:** Activities in the Online Consultant are executed in parallel and any number of times, unless constrained by certain constraints to be described below. The state of the Online Consultant records whether an activity has been executed or not. If an activity has been executed, its state can be reset under certain circumstances to be described below. There are the following pre-defined activity types in the Online Consultant:

*eForm Activity:* As described above, the principal activity of the Online Consultant is the eForm activity. Each activity of this type has exactly one eForm attached to it, which is displayed to the user when the activity is executed.

*Invitation Activity:* This type of activity attaches a role to an external user (identified by an email address) and sends him an invitation link to the process instance via email notification.

*Signing Activity:* This type of activity is used when the data input by the end users needs to be digitally signed using digital certificates. The input data from multiple eForms can be signed in a single signing activity.

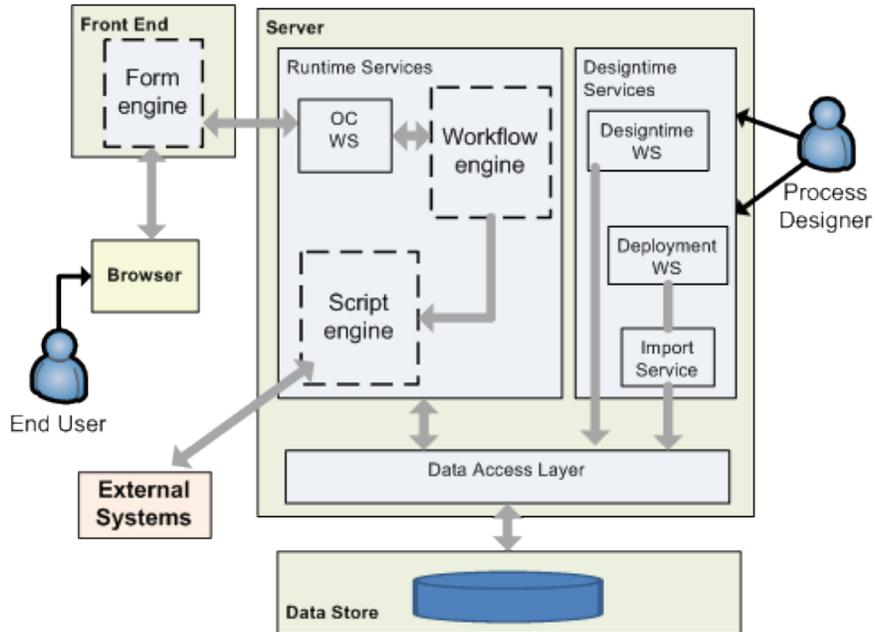


Fig. 1. The Online Consultant Architecture.

*External Activity:* Via a general script engine it is possible to connect to any external system, e.g. for automated tasks.

We will only consider eForm activities in the remainder of the paper.

**Transactions:** An Online Consultant Transaction holds a group of activities to be executed in transaction mode. The transactions differ from standard transactional semantics in that they are long running and cannot be rolled back. Instead, as also found in web-service orchestration languages such as WS-BPEL, they can have a compensating logic to be executed in case a transaction has to be aborted. Transactions can be either signed or unsigned. Signed transactions involves signing the data using digital certificates by single/multiple parties containing many eForms. We will leave the further investigation and formalization of transactions for future work.

**Resources/Roles:** The Online Consultant workflow has a simple resource model that uses Roles to define allowed behaviour of different users within the system. Each Role is assigned an access right for each activity of a workflow. The possible access rights are Read (R), Write (W) and Denied (D). The Read access right allows a user with the particular role to see the data of an activity, where as Write access right allows the user to execute an activity and also to input and submit data for that activity. A Denied access right has the effect of making the activity invisible to the user.

**Control Flow Primitives:** The Online Consultant workflow contains the following control flow primitives which control the activity execution at runtime.

*Activity Condition:* Every activity in the Online Consultant has an attached activity condition. An activity condition is a boolean expression that can reference the variables from the shared data store. When this condition evaluates to true, the activity is included in the workflow for execution, on the other hand the activity will be skipped from the list of activities stacked for execution if the condition evaluates to false. Activity Conditions are re-evaluated whenever necessary, so the inclusion of an activity in the workflow can be changed within in the lifetime of the workflow instance.

*Sequential Predecessors:* A sequential predecessor is a constraint that states that if activity B has a sequential predecessor activity A, then B can execute, when once A has finished executing. However, the sequential predecessor has only effect if the predecessor activity A is included in the workflow as per its activity condition. Suppose the predecessor activity A is not part of the workflow at runtime at a certain point of time, then activity B can be executed before A is executed even if A is a sequential predecessor of B. If activity A becomes part of the workflow runtime after B has executed, because of some runtime state changes which might have affected the activity condition for activity A, then activity A will be executed and this will not affect activity B.

The central idea of the Process Matrix is that only the *necessary* constraints between activities are declared and that decisions on whether an activity should be included in the process or not are separated from the control flow. Activity conditions and sequential predecessor constraints allow an activity to be carried out at any time and any number of times, as long as its activity condition evaluates to true and all of the required predecessors (for which the activity condition evaluates to true) are carried out. This is not always the intended behavior, so a stricter notion of predecessor is introduced, called *logical predecessor*.

*Logical Predecessors:* If activity B has activity A as a *logical* predecessor, then A is a sequential predecessor of B, but in addition, if activity A is re-executed, then activity B must be re-executed. Also, if the state of activity A is reset (as described below), and activity B is included, then B is also reset and cannot execute again until activity A has been executed again. Like a sequential predecessor, a logical predecessor is only evaluated if the predecessor activity is part of the workflow at the time of execution. However, if an activity A which is logical predecessor of B becomes part of the workflow after activity B has been executed, the activity B will be reset and hence the activity B must be executed once again.

In order to understand logical predecessor constraints better, let us consider two activities in a workflow, activity A (an eForm Activity) and activity B (a Signing Activity). If the purpose of activity B is to digitally sign the user data in activity A, we may choose to make A a logical predecessor for B: In case when activity A has been re-executed (and the data typed into the form potentially modified, then activity B (signing the data) should be performed again.

The above information is collected in a matrix as shown in Fig. 2, which is referred to as the Process Matrix. Each row of the matrix represents an activity of the process. The columns are separated in 3 parts: The first set of columns

Steps	Roles			Predecessors	Activity Condition
	I	II	III		
step 1	R	W	D		
step 2	W	R	D	1	
step 3	R	W	R		$A \wedge B$
step 4	R	W	R	* 2, 3	$A \vee B$

**Fig. 2.** The Process Matrix.

describes the access rights for the different roles (role I, role II and role III in the figure). For instance, the *R* in the row of step 1 and column of role I indicate that users with role I has read access right, i.e. they can read the content of the form of step 1 but not enter data. The next row describes the predecessor constraints, where we indicate by a \* that the predecessor is a logical predecessor. That is, step 2 has step 1 as sequential predecessor and step 4 has step 2 as logical predecessor and step 3 as sequential predecessor. Finally, the last row describes the activity condition. For instance, the condition  $A \wedge B$  of step 3 indicates that the boolean values *A* and *B* in the shared data store must both be set to true for this activity to be included in the flow.

Boolean variables used in activity conditions are referred to as *purposes*. The reason for this terminology is that a boolean variable used in an activity condition in effect specify two *variants* of the workflow: If it evaluates to true then all activities where it is required in the activity condition is included, if it is false they are excluded. This makes it very easy to reuse a process description for a different purpose in a different variant: One just adds a new purpose variable and use it to toggle the relevant activities.

	Steps	Roles			Predecessors	Activity Condition
		App	CW	Mgr		
1	Application	W	R	R		
2	Register Customer Info	W	W	W		
3	Approval 1	D	W	R	* 1,2	
4	Approval 2	D	R	W	* 1,2	$\neg Rich$
5	Payment	R	W	R	* 3,4	$\neg Hurry \wedge Accept$
6	Express Payment	R	W	R	* 3,4	$Hurry \wedge Accept$
7	Rejection	R	W	R	* 3,4	$\neg Accept$
8	Archive	D	W	R	* 5,6,7	

**Fig. 3.** Process matrix for a loan application process.

Let us consider a concrete (toy) example of a workflow for loan applications shown in Fig. 3.<sup>1</sup>The example uses three roles: The *Applicant* (App), the *Case Worker* (CW) and the *Manager* (Mgr). The activities are: Filling in the application (Application), Registering customer information(Register Customer Info), Approval of the application (Approval 1 and 2), Payment, Express Payment, Rejection and Archive.

The Roles columns indicate that the applicant can fill out applications, but the case worker and manager can only read the content of the application. Everyone can register customer information, but only the case worker can perform approval 1 and only the manager can perform approval 2, and both approval steps are invisible to the applicant. The remaining actions can only be performed by the case worker - they can be read by the manager and applicant, except for the archiving which is invisible to the applicant.

The Activity Conditions in the last column depend on the purposes *Rich*, *Hurry* and *Accept*. A rich applicant only needs an approval from the case worker, while a poor applicant also needs an approval from the manager in the bank. If the purpose *Hurry* is set to true, the application is treated as an express payment. The result is that the Express payment activity (step 6) is included and not the Payment activity (step 5). Conversely, if the purpose *Hurry* is set to false, the the (normal) Payment activity in step 5 is included and not the express payment activity. Both payment activities require the purpose *Accept* to be true.

	Steps	Activity Condition	Activity Status
1	Application	true	executed
2	Register Customer Info	true	executed
3	Approval 1	true	can start
4	Approval 2	true ( $\neg Rich$ )	executed
5	Payment	true	can not start (wait for {3})
6	Express Payment	false( $\neg Hurry$ )	inactive ( $\neg Hurry$ )
7	Rejection	true	can not start (wait for {3} $\wedge$ $\neg Accept$ )
8	Archive	true	can not start (wait for ({3} $\wedge$ {4}) $\vee$ ({3} $\wedge$ {7}))

**Fig. 4.** The Process Matrix at Run Time.

Finally, the Predecessors column specify that the approval actions require that the application and customer info actions has been carried out - and by making application a logical predecessor we enforce the approval to be carried out again if the application is re-filled. To allow for more fine-grained constraints, the Process Matrix model includes an additional advanced feature called *dependency expressions*. Dependency expressions are a set of expressions attached to

<sup>1</sup> The example is available for online demonstration at ([www.resultmaker.com](http://www.resultmaker.com)).

an activity. Like activity conditions, dependency expressions can also contain references to variables in the shared store. However, where Activity Condition evaluates to boolean values, dependency expression can evaluate to any value. Any change in the value of the dependency expression will reset the activity state to indicate that the activity has not been executed. For instance, it may be that only changes in the amount, typed into the eForm attached to the Application step, in the above example should cause approval to be re-executed. Then the amount of the loan can be made a dependency expression and the application can be made a normal sequential predecessor of the approval steps.

In Fig. 4 we show a possible state of the system during an instance of the workflow where a poor applicant applies for a non-express loan. The activity condition for all other activities except activity Express Payment is set to true and they are included for execution. The activity Approval 2 is also included because of the purpose *Rich* has value equals to false. The purpose *Hurry* is set to false and this makes the activity status of activity Express Payment to inactive. The activities Application, Register Customer Info, Approval 2 have already been executed and their activity status is set to executed. The activity Approval 1 ready for execution, but it has not started executing with a status of can start. Note that the activities Payment and Rejection can not be started because of their predecessors, but only one of them will be executed in future as the value of purpose *Accept* makes the other activity to be excluded from the list because of its activity condition. The activity Archive will be executed eventually after all its predecessors, as it does not have any purposes attached to it. As mentioned before Activity conditions will be re-evaluated after execution of each activity which makes the dynamic inclusion or exclusion of activities possible at runtime.

Note that it is nowhere specified when the values of the purposes are set. This is essentially left to the definition of the eForms for the individual activities.<sup>2</sup> Also note that the registration of customer information can be done either before or after the application, and can be redone arbitrarily often without affecting any of the other steps.

We conclude the presentation of the Process Matrix by comparing it to a typical flow chart process notation. Fig. 5 shows a possible description of the loan process as a flow chart. One immediately notice that it assumes that every activity is carried out once. Also, Approval 1 is carried out first (since it is always needed) - but this forces an order on the two approval steps. The Process Matrix model allows the activities to be executed in any order or even in parallel, as there are no constraints between them. Clearly, changing the flow chart diagram to allow every possible path of Process Matrix description would make it much more complex.

---

<sup>2</sup> In the online example at ([www.resultmaker.com](http://www.resultmaker.com)) the purposes *Rich* and *Hurry* are set (by radio buttons) in the eForm attached to the Application activity in step 1, and the purpose *Accept* is toggled in the eForms attached to Approval 1 and Approval 2.

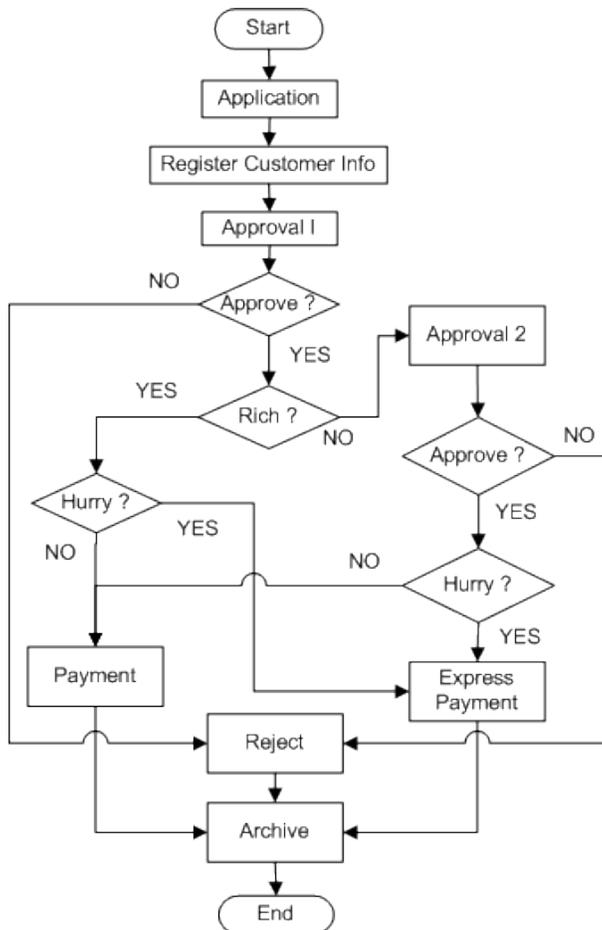


Fig. 5. Example in Flow chart.

### 3 Case story: The Digital Building Permit

In this section we describe a real case story on the use of the Online Consultant for digitalizing the rules for building permits in one of the largest municipalities in Denmark.

#### 3.1 The digital building permit in the municipality Frederiksberg

Frederiksberg is one of Denmark's largest municipalities. Each year it handles about 2100 building permit applications that vary in type and complexity. Most of these applications are for relatively minor modifications of apartments—for example, renovation of a bathroom or lavatory, combining two apartments into one, etc.—and are made by ordinary citizens who are not professionally engaged in the building trades.

The permits and their application process is complex, regulated by 2 main areas of regulation, each subdivided in sub areas which have many variants depending on the nature of the construction / modification project. Furthermore the process of applying depends on the relationship of the person(s) that inhabit the building or apartment in question with the owner(s), as well as the number of neighbors that need to be involved as well as the number of experts (architects etc.) that need to be involved. The number of enclosures such as drawings, declarations etc, that are needed to be included, depend on the nature of the application, the regulation and some other variants. Also the documents or enclosures are needed to be signed by Multiple parties.

Until recently, paper forms were used for submitting building permit applications, and in many cases, the permits were not correctly filled out. Enclosures might be missing or they might be incomplete in relation to the relevant type of project.

Building regulations are so extensive and complex that it can be difficult to figure out what information, documents and powers of attorney the municipality will need in order to process the application. In practice, about 70% of building permit applications are missing some of the necessary information when they arrive at the municipal offices, which mean that they cannot be processed right away. A large part of case workers time is therefore used for corresponding with citizens to acquire the information needed for processing their applications, instead of doing the actual processing.

The overall number of variants is huge in the building permit process, and also the variant that ends up being selected depends on the situation as it evolves during the application process. Some times the assumptions may change during the application process due to a declaration or expert statement that differs from the initial assumptions, making the process to flow back.

The municipality therefore needed a digital solution that would guide citizens through the steps of a building permit application and ensure that all the relevant data, enclosures and powers of attorney were correctly formulated and added to the application before it was submitted to the municipality.

Resultmaker developed the Digital Building Permit based on Resultmaker Online Consultant. The solution handles the complete flow of steps for a building permit application, from the citizens typing in data and acquiring powers of attorney before submitting the application, to a message of completion when the construction work is finished.

By modeling the building permit application process in the Process Matrix each regulation element and each variant condition became decoupled from each other, and when the matrix joins them up, the result is a correct compilation of flow conditions, that may hold all variants while staying compliant with regulation. If a traditional flow chart approach were adopted, the process would either have been unacceptably rigid for the users, or it would explode in complexity and become prohibitive to develop let alone maintain when updates to the regulation is passed in parliament.

The architecture of the solution ensures that it can easily be expanded both in relation to the existing types of building permits, in relation to possibly adding new types, such as rural or commercial buildings and also flexible enough to absorb regulation updates.

The result of the digital solution is that the application process has become simpler and more convenient for citizens and more efficient for the municipality, and cases are handled more quickly. Citizens can submit an application without prior knowledge of the relevant laws and regulations. All they have to do is state what kind of work they want to have done for example, replace a toilet, move a wall, etc. Digital Building Permit figures out the necessary steps and tells the applicant which drawings, permits and other enclosures are required. The time expenditure for checking, following up on missing information and the number of errors in application process are minimized. Case workers can concentrate on case handling, which is carried out faster and more reliable.

## 4 Formalizing the Online Consultant in LTL

In this section we follow the approach in [9, 8] and formalize the key primitives of the Online Consultant process matrix described in Sec. 2 in Linear Time Temporal Logic (LTL) [5, 6]. In particular, we relate the formalization to the constraint templates employed in the DecSerFlow and ConDec languages described in [9, 8].

### 4.1 LTL and DecSerFlow

LTL is a temporal logic for describing (infinite) sequences of steps. Similarly to [9, 8] we assume a discrete time model where any step in the sequence corresponds to the execution of one activity.<sup>3</sup>

The basic propositional formulas of the logic are boolean formulas over propositions on the state space and the current activity, in particular we assume

---

<sup>3</sup> To deal with the fact that LTL really is interpreted over infinite sequences we follow [9, 8] and assume that finite executions are terminated by an infinite sequence of steps with no activity and no change in the store.

propositions on the form (**act** ==  $A$ ) to determine the current activity, i.e. the proposition (**act** ==  $A$ ) is true if the current activity being executed is  $A$ .

The basic propositional formulas are extended to sequences using the *temporal* modal operators  $\mathbf{O}P$  (in the next state of the sequence formula  $P$  holds),  $\mathbf{\square}P$  (in the current and all of the following states of the sequence formula  $P$  holds),  $\mathbf{\diamond}P$  (in the current or at least one of the following states of the sequence formula  $P$  holds), and  $Q \mathbf{U} P$  (in the current or at least one of the following states of the sequence formula  $P$  holds and formula  $Q$  holds in all states *until* that state is reached). Note that  $\mathbf{\diamond}P$  could be written as  $\mathbf{True} \mathbf{U} P$ .

By using a logic one can specify the sequences in a *declarative way* by the *properties* they must satisfy, e.g. that a certain event must always occur before another event. This should be seen in contrast to *imperative* specifications, such as automata or flow diagrams, specifying *how* the events are occurring. The difference is thus on the *ease* (or compactness) of expression and not on expressiveness: One can express exactly the same sets of valid sequences in LTL as one can express in the automaton model for infinite sequences [6]. LTL has been extensively used as property language for automatic verification of reactive systems, also referred as *model checking* [4]. The basic principle of model checking is that an intended property of a system is declared in a specification language, such as LTL, and an automatic tool then checks if a system, e.g. specified by an automaton, satisfies the property. In this case one say that the system is a *model* of the property. The key idea of the approach in [9] is to turn this around and use the declarative specification language as system definition. One may then use the correspondence between the logic and automata to construct an automata that can be used for execution of the process. Alternatively (and probably more efficient) one may use techniques from on-the-fly/dynamic model checking to dynamically construct from the formula and partial execution trace the next possible actions. It is important to note that the formulas specify what should hold for the *completed* sequence. That is, a partial execution sequence need not satisfy the formula to be valid, as long as it is possible to complete the sequence in a way that makes the formula satisfied.

In acknowledgement to the fact that LTL formulas may be too difficult to understand for process designers, the approach in [9] proposes to use so-called *constraint template formulas*, also referred to as policies or business rules. These templates are then equipped with a graphical notation for relations between activities drawn as boxes.

A basic example of a template in the DecSerFlow language is the constraint template *existence*( $A : activity$ ) formalized as  $\mathbf{\diamond}(\mathbf{act} == A)$  in LTL. It simply states that there exists a step in which activity  $A$  is carried out.

An example of a so-called *relation formula* [9] is the constraint *precedence*( $A : activity, B : activity$ ) which states that an activity  $B$  is preceded by an activity  $A$ , i.e. the activity  $B$  can not be executed before activity  $A$  has been executed. This template formula is expressed in LTL as

$$existence(B) \implies (!(\mathbf{act} == B) \mathbf{U} (\mathbf{act} == A))$$

where  $!$  denote the the boolean negation. Reading the formula, it expresses that if there exists a step in which  $B$  is carried out then there exists a step in which  $A$  is carried out, for which  $B$  is not carried out in any of the preceding steps. This is equivalent to that the activity  $B$  can not be executed before activity  $A$  has been executed as wished. Note that the template uses the existence template as a sub formula.

Another example of a relation formula is the constraint  $response(A : activity, B : activity)$  which express that if whenever the activity  $A$  is executed then  $B$  must also be executed after it. This formula is expressed as

$$\Box((\mathbf{act} == A) \implies existence(B))$$

From the response and precedence templates one may build composite relation templates, such as the template  $succession(A : activity, B : activity)$  expressed in LTL simply as a conjunction of the two templates:

$$response(A, B) \wedge precedence(A, B)$$

The formula expresses that every execution of activity  $A$  must be followed by an execution of  $B$  and any execution of  $B$  must be preceded by an execution of  $A$ .

The reader may already have noticed similarities with the primitives in the Process Matrix. In the following section we will see that the Process Matrix primitives can indeed be formalized similarly to the templates given above, but leads to interesting variations due to the use of activity and dependency conditions. We do not consider the roles.

## 4.2 Formalization of the Process Matrix

The formalization of a Process Matrix workflow will be a set of formulas in conjunction. We let  $A$  and  $B$  range over activities (steps) in the process model and write  $actcon(A)$  for the activity condition of activity  $A$ .

To keep the formalization simple we will assume that the state of the steps in the sequences records the execution status of each activity as described in Sec 2, and that we have basic propositions  $executed(A)$  and  $reset(A)$  that reflects if an activity  $A$  has activity status executed or reset respectively. If there is a chain of logical predecessors  $A_0 \overset{*}{<} A_1 \overset{*}{<} \dots \overset{*}{<} A_k$  which in a state all have status of executed and are all included (i.e.  $actcon(A_i)$  is true for  $i \in \{0, \dots, k\}$ ) and the first activity  $A_0$  is reset (i.e. a dependency expression is changed or the activity is re-executed by the user) then the logical successors  $A_i$  for  $i \in \{1, \dots, k\}$  will also be reset in the Process Matrix. The same happens in the situation above if  $A_0$  is included into the workflow (i.e. changing  $actcon(A_0)$  to true). The two basic propositions  $executed(A)$  and  $reset(A)$  may be expressed in LTL as part of the formulas, thereby avoiding the requirement that the steps record the execution status. However, it makes the formalization far more complex.

The first formula used for the formalization is the LTL formula  $act\_include(A : activity)$  given by

$$(\mathbf{act} == A) \implies actcon(A)$$

It expresses that an activity  $A$  can only be executed if it is included, i.e. its activity condition is true. The formula is included for every activity in a workflow.

We then define two templates used as sub formulas in the formalization of the Process Matrix primitives. The first such template is  $act\_including(A, B)$  which expresses that activity  $A$  is executed and at the same time the activity  $B$  is included in the process (because the activity condition for  $B$  is true). This is formalized in LTL as:

$$(\mathbf{act} == A) \wedge actcon(B)$$

The second sub formula template is  $existence\_act\_including(A, B)$  which extends the existence template for DecSerFlow to express that an activity  $A$  is eventually executed and at the same time the activity  $B$  is included in the process. This is formalized in LTL as:

$$\diamond act\_including(A, B)$$

We now go on to formalize the primitives of the Process Matrix.

**Sequential Predecessor:** The sequential predecessor constraint is similar to the precedence formula in DecSerFlow described above, except for the use of the activity condition in the Process Matrix. We define the constraint template  $sequential\_predecessor(A : activity, B : activity)$  stating that  $A$  is a sequential predecessor of  $B$  by the LTL formula

$$existence\_act\_including(B, A) \implies (!act\_including(B, A) \mathbf{U} (\mathbf{act} == A))$$

**Logical Predecessor:** Logical Predecessor is a strengthening of the Sequential Predecessor constraint. The basic proposition  $reset(A)$  allow us to formalize the template  $logical\_predecessor(A : activity, B : activity)$  in LTL as  $sequential\_predecessor(A, B)$  in conjunction with

$$\square(reset(A) \implies sequential\_predecessor(A : activity, B : activity))$$

**Activity Execution:** The final part of the formalization, is to express when an activity should be executed. Benefitting from the assumed basic proposition  $executed(A)$  the execution property can be formalized as

$$(\square \diamond executed(A)) \vee (\square \diamond !actcon(A))$$

which is included for every activity  $A$ . The formula expresses that the activity  $A$  either infinitely often has the status executed or it is infinitely often excluded from the process.

## 5 Conclusion and Future Work

We have presented the patented process model employed in the Resultmaker, On-line Consultant (OC) workflow management system as an example of a flexible

declarative workflow process model used in practice. We described and formalize the key primitives of the OC process model (sequential predecessor, logical predecessor, activity conditions and dependency conditions) as LTL formulas. This supports the recently proposed approach in [9, 8] to use LTL as the foundation for flexible declarative process languages. The formalization suggested interesting new constraint templates for the ConDec and DecSerFlow languages based on this approach, that take account for dynamic changes in activity and dependency conditions. On the other hand, the formalization also suggest extensions to the Online Consultant to be considered in future work: In particular one may consider extensions to the primitives of the Process Matrix, or even follow the general approach in DecSerFlow and allow designers to specify LTL formulas, using templates presented in a graphical form as in the DecSerFlow language.

In being a declarative language, LTL and thus DecSerFlow are well suited as languages for formalizing the primitives of the Process Matrix. The use of LTL in model checking may be exploited for the implementation and verification of workflow processes specified in LTL. However, one should be aware that other temporal logics for specifying computations exists, notably the *Computational Tree Logics* CTL\* and CTL [4]. We leave for future work the possible uses of branching time logics and primitives derived from such logics for specifying workflow processes.

Future work will also include studies of the use of the Online Consultant Process Matrix for pervasive healthcare services. Indeed, flexible clinical guidelines are right now being implemented using the Process Matrix as part of a Polish Electronic Health Record project (EHR-PL). As part of the TrustCare project [3] we will research extensions of the Process Matrix to pervasive adaptable workflow processes based on formalizations of the process model.

## References

- [1] Petra Heintl, Stefan Horn, Stefan Jablonski, Jens Neeb, Katrin Stein, and Michael Teschke. A comprehensive approach to flexibility in workflow management systems. In *In Proceedings of WACC '99*, pages 79–88. ACM Press, 1999.
- [2] Thomas Hildebrandt. Computer supported mobile adaptive business processes (CosmoBiz) research project. Webpage, 2007. (<http://www.cosmobiz.org/>).
- [3] Thomas Hildebrandt. Trustworthy pervasive healthcare processes (TrustCare) research project. Webpage, 2008. (<http://www.trustcare.dk/>).
- [4] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. Lucent Technologies, 1999.
- [5] A. Pnueli. The temporal logic of programs. In *In Proceedings of 18th IEEE FOCS*, pages 46–57, 1977.
- [6] A.P. Sistla, M. Vardi, and P. Wolper. Reasoning about infinite computation paths. In *In Proceedings of 24th IEEE FOCS*, pages 185–194, 1983.
- [7] W M P van der Aalst and S Jablonski. Dealing with workflow change: identification of issues and solutions. *International Journal of Computer Systems Science & Engineering*, 15(5):267–276, September 2000.
- [8] W.M.P van der Aalst and M. Pesic. A declarative approach for flexible business processes management. In *In Proceedings of Workshop on Dynamic Process Management (DPM 2006)*, volume 4103 of *LNCS*, pages 169–180. Springer Verlag, 2006.

- [9] W.M.P van der Aalst and M. Pesic. DecSerFlow: Towards a truly declarative service flow language. In M. Bravetti, M. Nunez, and Gianluigi Zavattaro, editors, *In Proceedings of Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *LNCS*, pages 1–23. Springer Verlag, 2006.