An Architecture for a Blended Workflow System

Bernardo Oliveira Pinto

Instituto Superior Técnico
bernardopinto@ist.utl.pt

1 Introduction

Most industrial workflow systems are activity-based. The work is specified by defining activities and on how they can be coordinated using control-flow primitives, as sequential and parallel execution, to achieve the business process goals \cite{1}. Activity-based workflows prescribe the activities execution order and lack flexibility to handle unexpected situations for which they were not codified.

In a different trend, closely related to knowledge work, both researchers and industry are proposing new workflow approaches that foster users collaboration to deal with unexpected situations. These workflows support ad-hoc behaviour and delegate to end users the responsibility to guarantee that the business process goals are achieved. However, they lack the guidance provided by activity-based workflows.

The blended workflow approach \cite{2} is a new approach which intends to bridge the gap between completely structured workflows and ad-hoc workflows. The idea behind the blended workflow approach is that a workflow management system should allow users to deviate from the structured execution whenever it is necessary, but may allow them to regain the guidance provided by structured workflows once the unexpected situation is dealt with. To do so, blended workflow proposes the consistent coexistence of two workflow models, a prescriptive activity-based model and a descriptive goal-based model.

Current workflow engines either support prescriptive workflows \cite{3} or descriptive ones \cite{4}, but, as far as we know, there is no proposal for a workflow engine that integrates both. The challenges that such engine has to face are on how to support the consistent execution of both models, such that during execution users can move back and forth between these two perspectives.

In this paper we will describe a solution to the implementation of this approach, i.e. how can we join activities and goals in a workflow management system, in such a manner that the system behaves as described in the blended workflow approach \cite{2}. Actually, a third model, an object model, is defined to integrate the execution of the goal and activity models.

In the next section we describe the architecture of the solution driven by its relevant aspects, and section 3 uses an example to show how the workflow prototype can be used. The current prototype is described in section 4 and we drive some conclusions of our work, in section 5.
2 Blended Workflow System Architecture

The architecture of the blended workflow system follows the Workflow Reference Model \[5\]. A central service of the Workflow Reference Model is the Workflow Enactment Service, which is responsible for interpreting the process specification and for executing process instances.

The Workflow Enactment Service of the blended workflow system architecture is composed by two Workflow Engines, depicted in figure 1. The Activity Workflow Engine is responsible for activity management whereas the Goal Workflow Engine is responsible for goal management. These two engines provide the end user with two independent perspectives of the workflow instance. Although each end user action is applied to only one of the engines, both perspectives are updated to give consistent views of the workflow instance.

![Workflow diagram](image)

Fig. 1. The general architecture of the blended workflow system.

The user interacts with the Activity Engine and the Goal Engine. The engines are synchronised through a shared Data Repository. This repository contains the data that is used by both engines and ensures that they always access the most up-to-date data. Therefore, a notification mechanism is used to inform engines about changes. Both engines are always notified because the changes in the data model may impact on more activities and/or goals than the one manipulated by the end user action, as it will be explained below.

The Activity Workflow Engine comprises the activities and the control-flow between them. In addition, the engine execution extends a traditional activity-based workflow engine by including specific pre- and post-conditions. To enable an activity for execution, its pre-condition must hold true, and an activity is considered executed if its post-condition holds true. The activity engine of a blended workflow integrates the traditional control-flow with an additional data-flow, given by the pre- and post-conditions. The activities pre- and post-conditions are used by the activity engine to evaluate the shared state, represented by the
dashed arrows from conditions to the shared state, and conclude about which activities can be enabled or are completed.

The Goal Workflow Engine is constituted by goals and goal decomposition relationships. Goals are specified by a condition over the data in the Data Repository. End users interactions with the goal engine result in changes in shared state that fulfil goals, i.e. their conditions hold true.

A blended workflow instance has to execute according to both activity-based and goal-based specifications. This means that any successful execution following the activity-based specification must fulfil the goal specification, and in particular the top goal. The goal conditions and the activities pre- and post-conditions constitute the blended workflow specification for goal achievement, whereas the activities control-flow specify a particular behaviour, the standard behaviour, on how to achieve the goals. Therefore, the activity-based specification is an overspecification of the goal specification and the conditions over the Data Repository is where both specifications overlap.

The Data Repository contains the data entities, their attributes, and the relationships among them. During execution of a blended workflow instance, both engines change the shared state and the execution completes when the shared state makes the goal conditions hold true.

End users interact with workflow instances through both interfaces. In addition to the execute activity and fulfil goal operations, three other operations provide further execution flexibility: skip activity, skip goal and create goal. Skip activity operations allow end users to leave out an activity in the execution of a workflow instance, because there was a change in the conditions that permit its execution, for instance the actor cannot perform the work. Similarly, skip goal operations allow end users to disregard the execution of a goal, possibly because of an unexpected situation the goal became nonessential for the workflow instance. Finally, operation create goal empower end users to define new goals for a workflow instance.

We describe how the blended workflow system architecture supports operations: activity execution, skip activity, goal achievement, skip goal, and create goal.

Activity execution. For a user to execute an activity it has to be enabled by control-flow and its pre-condition must hold true, this means the data in the data repository is in a state that satisfies the activity’s pre-condition. After activity execution, its post-condition holds true, which means the data in the data repository is in a state that satisfies the activity’s post-condition. However, there may be the case where the activity’s pre-condition does not hold true, because it refers data that was not produced by any other previous activity (see below how pre-activities can be generated to deal with this situation). When there are changes in the shared state, the conditions that depend on the updated data are re-evaluated.

Skip activity. When an activity is skipped, the state of the data referred by the activity post-condition is changed to skipped and the control flow proceeds as if the activity has been executed, yet its post-condition is not fulfilled. Similarly
to activity execution, the change in the data state triggers the conditions that depend on the changed data.

**Goal achievement.** When a goal is achieved, the blended workflow evaluates which activities may have complete.

**Skip goal.** When a goal is skipped, all its sub-goals, that were not achieved yet, are also skipped and the state of the data referred by these goals’ definition is also changed to skipped. The activity engine re-evaluates the post condition of all the activities enabled by control-flow and if the result of the evaluation is skipped, the activities are skipped. Afterwards, the blended workflow analyses the pre-conditions of the activities which are enabled by control-flow, and depend on the skipped data, and generate pre-activities for them.

**Create goal.** Users can create a new goal for a particular workflow instance. By creating a new goal, they also have to specify the condition that defines it. Additionally, the user needs to create new entities and/or add new attributes to existing entities, which the condition refers. A new goal achievement does not have any impact on the activity-based specification.

Pre-activities are generated when there is a need to enable for execution an activity that is enabled by control-flow but its pre-condition does not hold true. This happens when the user skips a previous activity or a goal. The system generates pre-activities that empower the end user provide the missing data and enable the activity for execution.

3 Application Example

To illustrate the blended workflow approach consider an example of a medical episode. The activity-based specification of medical episode comprises the activities Check-in Patient, Collect Data, Physical Examination, Doctor Appointment and Check-out Patient. The goal specification defines a set of goals: Diagnose Patient, Observe Patient, Write Medical Report, Collect Data, Physical Examination and Prescribe.

These two specifications use the data specification that is presented in Figure 2.

![Figure 2](image-url)
As examples of the conditions implemented in the activity and goal specification, consider the Medical Report entity. The condition of the Write Medical Report goal requires the creation of a Medical Report object containing a written diagnosis written by the doctor. In the activity-based view, Doctor Appointment post-condition also requires the creation of the Medical Report object and, optionally, a Prescription object can also be created. Therefore, Write Medical Report can be achieved without requiring the optional goal Prescribe to be fulfilled. On the other hand, Doctor Appointment activity pre-condition requires the existence of object Patient Data, holding values for attributes Height, Weight, and Physical Examination, on which post-conditions of activities Collect Data and Physical Examination are dependent.

To illustrate the flexibility of the blended workflow approach we describe an unexpected situation, an administrative strike.

In the case of an administrative strike, the activities performed by the administrative staff cannot be executed. There are two ways of executing the workflow in this case:

**Skipping activities and executing pre-activities.** In this case, the nurses have to skip activity Check-in Patient.

Next, they will also have to skip activity Collect data and execute a pre-activity to enable Physical Examination activity. This pre-activity consists in defining Patient and Episode objects, which are required by Physical Examination pre-condition. Once Physical Examination activity is executed, both the activity and the goal are, respectively, executed and accomplished.

Finally, the physician has to execute a pre-activity to set the Height and Weight attributes and thus, enable the Doctor Appointment activity. After executing this pre-activity, the goal Collect Data is fulfilled and the the activity Doctor Appointment becomes executable again. After the physician executes Doctor Appointment activity the goal Diagnose Patient is fulfilled.

**Skipping activities and achieving goals.** To do this deviation from the “usual” execution, the physician only needs to achieve the top goal. To do so, he has to skip the first three activities (Check-in Patient, Collect Data and Physical Examination), and then, he can achieve the Diagnose Patient goal by achieving its mandatory subgoals, Observe Patient and Write Medical Report. In this case, only the Doctor Appointment activity will be considered as completed, since its post-condition is satisfied by Write Medical Report achievement.

### 4 System Implementation

The system implementation addresses the issue of how to keep the activity and goal views consistent, so when an activity is executed or a goal achieved the changes made in the data model are automatically reflected in both views. Before we describe the implementation of activities and goals, we first address the implementation of the data and conditions, because views consistency is built on top of them.
4.1 Data Implementation

The data model describes which are the entities, the attributes within each entity and the relations between entities. Moreover, it also specifies which are the key attributes/entities necessary for an entity to exist. This means that an entity is only defined when all its key attributes/entities are defined. The class diagram in UML of the data implementation is depicted in Figure 3. It follows a meta-model approach to allow the dynamic definition of new data entities, relations and attributes, which may be necessary when a new goal is created.

![Class diagram in UML of the data model.](image)

The `Entity` class represents an entity type in the data model specification. This class has a list of `Attribute` and a list of `Relation` it is part of. Relations are binary and have cardinalities. Class `EntityInstance` represents the instance of the entity and has a list of `AttributeInstance` and a list of `RelationInstance`. When an attribute instance is created its state is set to `empty`. An entity is in state `empty` when at least one of its key attributes is in state `empty` or one of its key relations refer to an empty entity. After an attribute instance value is set, its state is updated to `defined`.

When a blended workflow specification is loaded, instances of `Entity`, `Attribute` and `Relation` are created, and for each new workflow instance the shared space is populated with the minimal set of instances of `EntityInstance`, `AttributeInstance` and `RelationInstance` required to achieve the workflow goals. Afterwards, during the workflow execution, the values of the attribute instances are set and, eventually, the goals will be achieved.

Since the blended workflow allows the definition of a new goal during workflow execution, new objects of `Entity`, `Attribute` and `Relation` can be created for a particular workflow instance. In this case, these objects are private for the particular workflow instance.

Considering the medical episode example and its data model, depicted in figure 2 when the process is defined, one instance of the `Entity` class is created for each one of the entities in the data model: `Patient`, `Episode`, `Patient Data`, `Prescription` and `Medical Report`. Instances of `Attribute` are also created for each attribute, e.g. along with the entity `Medical Report`, an instance of the `Attribute` is created for the `Report` attribute and another for the `Closed` attribute. Being the `Report` the `Medical Report`’s key attribute, its `keyAttribute`
attribute is set to true. Along with the entity and attribute creation, the system also creates an instance of the Relation class, for each relation between entities.

During the workflow execution it may be the case that the physician creates more than one Prescription. In this situation the system creates another instance of the EntityInstance associated to the Prescription object, along with the necessary AttributeInstance objects, and one RelationInstance object, which relates the existing EntityInstance of the EpisodeInstance and the new PrescriptionInstance.

4.2 Conditions Implementation

The activity pre- and post-conditions and the goal condition are all defined in the same way: as a composition of predicates. These predicates are atomic and refer to a data element, either an EntityInstance object or an AttributeInstance object. Conditions follow a three-valued logic with values: true, false and skipped. A condition implements the logical operators and, or and not, according to the truth tables in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>(a) Not</th>
<th>(b) And</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>AND</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>Skipped</td>
<td>Skipped</td>
</tr>
</tbody>
</table>

(b) And

<table>
<thead>
<tr>
<th></th>
<th>True</th>
<th>False</th>
<th>Skipped</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>Skipped</td>
<td>True</td>
<td>Skipped</td>
</tr>
</tbody>
</table>

Table 1. The values that the condition returns when applying the specified logical operators.

Each predicate is associated to only one activity or goal. We consider two types of atomic predicates: the Exists(data) predicate and a relational predicate Relational(data, value) predicate, which compares data with value. The return values for these predicates are presented in Table 2.

(a) Exists

<table>
<thead>
<tr>
<th></th>
<th>State</th>
<th>Defined</th>
<th>Empty</th>
<th>Skipped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eval.</td>
<td>True</td>
<td>False</td>
<td>Skipped</td>
<td></td>
</tr>
</tbody>
</table>

(b) Relational (==, >, <, ...)

<table>
<thead>
<tr>
<th></th>
<th>State</th>
<th>Defined</th>
<th>Empty</th>
<th>Skipped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eval.</td>
<td>True/False</td>
<td>False</td>
<td>True</td>
<td>Skipped</td>
</tr>
</tbody>
</table>

Table 2. The values that the predicates return.
To implement these conditions, we used the declarative style described in [6]. The relation among conditions, activities and goals, and the structure of the conditions’ implementation is depicted in figure 4.

These operators allow one to aggregate the predicates mentioned before and to specify a composition of predicates that are optional. Hence, considering, for example, the post-condition of the Collect data activity, it is specified as Exists(Patient Data.height).and(Exists(Patient Data.weight)) and the goal Physical Examination definition is specified as Exists(Patient Data.Physical Report).and(Exists(Patient Data.Physical Examination)). These conditions correspond to the composition of two ExistsCondition objects in the context of an AndCondition object.

4.3 Activities and goals implementation

As described in Section 2 when a data value is changed both views are notified. The elements in the data model and the conditions implement the Observer pattern [7], where the “observers” are the conditions and the “observables” are the entity and attribute instances of the data model. This way, when there are changes in the data model, i.e. some entity or attribute is defined, skipped or created, only the state of the activities and goals that include the conditions which refer the updated data are re-evaluated. Figure 5 represents how the observer pattern is implemented. Class Condition inherits from DataObserver and classes EntityInstance and AttributeInstance extend DataObservable.

Skip operation differs from the activity execution and goal achievement operations in the way the shared space is changed. When an activity or a goal is skipped, all the AttributeInstance objects referred by the, respectively, activity post-condition and goal condition, which state is empty, are updated to state skipped. Afterwards, when conditions are re-evaluated, using the truth tables in Table 1 activities and goals can become skipped if they evaluate as skipped.

The process of adding a goal is handled by the Goal Workflow Engine. Thus, all the data needed, i.e. the conditions and the data to which they refer, the goal
name, its place in the goal hierarchy, whether is mandatory or not, is submitted to it. It is up to the goal engine to create the needed objects in order to add the goal to the process instance, e.g. submit the necessary data to the conditions factory to create the goal definition.

It may also be the case that new data must be created. In this case, the Goal Workflow Engine submits the necessary data to the Data Repository in order for it to create the entities/attributes, integrate them in the data model and create the empty instances. This has to be done before creating the conditions for the goal definition, because the specific object references of the entity/attribute instances are needed for the condition creation.

4.4 Implementation evolution

The implementation described in this work was an evolutive process. We started by implementing a prototype, with POJO\(^1\) to have a deeper understanding of the problem in hands and to reach our main objective: to integrate and synchronise both engines.

We then studied a manner to decouple the engines from the rest of the system, so that we can integrate already existing engines. The conclusion from this study was a minimal API which the engines must provide in order for integration with a blended workflow system.

The final implementation of our system is integrated with the YAWL Engine\(^2\).

5 Conclusions

This work describes the implementation of the blended workflow approach, a semi-structured approach that combines two perspectives to describe a workflow: an activity perspective, which describes the workflow in a completely structured manner, and a goal perspective, which describes the workflow in a declarative manner.

Blended Workflow allows users to perform various operations during the process execution: activity execution, skip activity execution, goal fulfilment, skip goal

\(^1\) POJO stands for Plain Old Java Object
\(^2\) http://www.yawlfoundation.org/
fulfilment, create new goal. With these operations the user can easily deviate from the structured execution whenever it is necessary. Moreover, the system keeps both models consistent, through a Data Repository, which receives the data updates (through the various operations the system allows) and propagates them. This way, the user is capable of regaining the guidance provided by structured workflow once the unexpected situation is dealt with.

For a more detailed description of this work, please refer to [8].

Acknowledgements This work was supported by FCT (INESC-ID multi-annual funding) through the PIDDAC Program funds.

References