Blended Workflow: Introduction of the Skip Operation

André Pereira Rodrigues
Instituto Superior Técnico – Lisbon, Portugal
apereirarodrigues@tecnico.ulisboa.pt

***Abstract*— In recent years, flexibility has been one of the hottest topics in business process workflow management. The ability to skip over predefined work tasks represents a flexible operation. This paper explains how a business process management environment called Blended Workflow implements such operation. Blended Workflow provides an imperative and a declarative view of the workflow and permits the exchange of execution between the two views during runtime. Our skip implementation allows the execution of an imperative workflow to deviate from its predetermined specification. The skipped data can be later defined using the declarative view of the workflow. With this addition, users do not have to necessarily execute what is initially specified by the imperative workflow, while still being able to achieve the business process' final goal. Overall, the skip operation makes a stricter workflow specification more flexible.**

I. INTRODUCTION

Workflow systems provide an organized and modular way to manage resources, assign work tasks, monitor process progress and information, as well as reducing the potential of errors and rework in any given business process [1]. Their importance cannot be understated, and they are used in virtually any working field in today's society.

There's not a single workflow implementation capable of successfully managing all business processes. Some working fields are more volatile than others, thus requiring different needs in terms of workflow adoption. Because of this variation, two main categories of workflow types have emerged within the industry: imperative and declarative [2].

Imperative approaches tend to favor process standardization, making them more suitable for scenarios where there is not a great deal of change from one process instance to another. A car manufacturing process is an example of an imperative process, where the production of the same car is the expected output every time the process is executed. On the other hand, declarative approaches thrive in dynamic environments where change is welcomed and each process instance may differ depending on different factors. A common example of this approach is the healthcare industry, where the treatment of patients depends on each case [3].

This notion of change is strongly related to the concept of flexibility. A flexible workflow management system is a system capable of adapting the business process implementation to foreseen and unforeseen changes, as well as being able to cope with the variability of dynamic environments [4]. Thus, it is believed that declarative approaches promote flexibility since the workflow management system has to be prepared with tools to deal with different conditions from case to case.

Due to this dichotomy between imperative and declarative methodologies the Blended Workflow approach was proposed [5], [6]. It manages the coexistence of two models, an imperative and a declarative model, of the same business process, such that users can perform their work according to the perspective that better fits their current context.

In this paper, we enhance the existing Blended Workflow approach with a new flexible operation that allows the skipping of unwanted activities, making the workflow tool better equipped to manage undesired and unexpected situations. Thus, we can state the main goal of our work as providing a new Blended Workflow operation that allows the partial execution of a given business process specification that is semantically consistent with the foundations Blended Workflow already had established. This means that even when work tasks are allowed to be skipped, the system must guarantee that the minimal fundamental business process goals are achieved.

The next section explains the overall behavior of Blended Workflow, followed by section III, which describes the implementation of the skip operation. Section IV overviews other approaches to the skip mechanism, and section V presents the closing remarks.

II. BLENDED WORKFLOW

Blended Workflow is a workflow management tool that allows the design, control, execution, and auditing of business processes. It covers all phases of a typical business process lifecycle, from its inception to conclusion, allowing a fluid manner to perform work tasks by giving the user the ability to choose between a declarative and an imperative view of the workflow specification.

The overall design process of Blended Workflow is depicted in figure 1. Essentially, Blended Workflow works as follows: a formal specification of a given business process is provided using a domain-specific modeling language. Based on that specification, a data model of the process is created, describing the data that has to be produced by the system and the dependencies between the data elements. From the data model, a state model is automatically generated, which contains the conditions that need to hold by any data model instance, as well as the intermediate and the successful final states. The state model serves as the foundation for the next two automatically generated models: the activity model and goal model. The generation strategy is done such that it preserves the state model in the generated models to guarantee their consistency, i.e. their execution only goes through consistent intermediate states and it can achieve

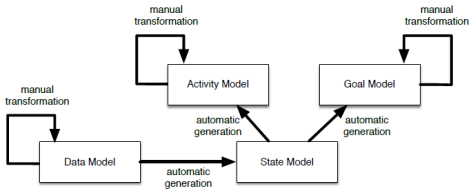


Fig. 1: Blended Workflow's design process. Image taken from [5].

a final state. After the automatic generation, the designer can apply a set of manual transformations to the generated models to enforce a particular behavior in the execution of the business process. For instance, in the activity model, by adding sequential dependencies it is possible to reduce the number of the possible sequences through which the business goal can be achieved, which results in the enforcement of a stricter standard behavior.

In the following sections, we will describe in more detail each of the main Blended Workflow process stages. We will use the organization of a university seminar scenario[1] as a working example of a business process applied to Blended Workflow. Organizing a university seminar usually comprises tasks such as deciding the seminar's title and topics, assigning staff members responsible for the seminar, enrolling students, and publishing their grades. We will see that by providing a rigorous data model, Blended Workflow will produce two drastically different workflows: one that is more focused on process flow – the activity model; and another one that provides more flexibility – the goal model. It is up to the end-user to use the preferred model, even though there's always the possibility to switch between them in runtime.

A. Data Model

The first step of the Blended Workflow business process implementation is to define the business process data model. The data models are mainly composed of entities, their attributes, the associations between entities, and explicit dependencies between attributes, which define a precedence order where the definition of an attribute requires the value of the attribute it depends on. Each attribute has a specific data type such as String, Number, or Boolean.

The data model of the organization of a university seminar scenario is pictured in figure 2. It is composed by entities `Seminar`, `Enrollment`, `StaffMember`, and `Topic`. Each entity is composed by a set of attributes like `title` and `description` in the case of the `Topic` entity. As we can see, the `Seminar` entity is mandatory which means that it has to be defined, along with all its attributes, to achieve the business process goal. In this data model, we assume that entity `StaffMember` was previously created and already exists in the system; the business process goal is about the seminar, not the hiring of staff members.

[1]Example inspired by [7].

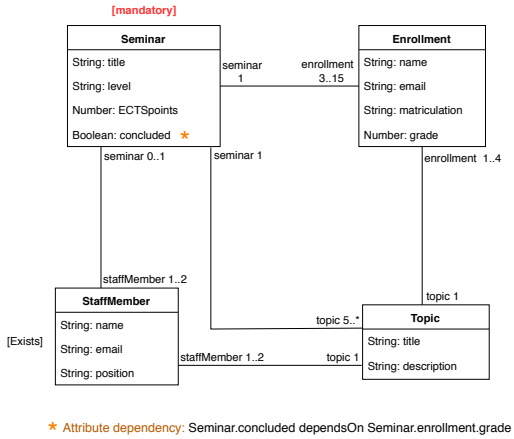


Fig. 2: Data model of the seminar organization scenario.

From this data model, we can tell that a given seminar can have one or two organizers (staff members), a minimum of five topics, and between three and fifteen students can be enrolled (represented by the `Enrollment` entity). Each topic has one or two supervisors (staff members) and can have from one to four assignees (students that enrolled on that specific topic). A student is enrolled in a seminar and may choose one topic from that seminar. A staff member is a supervisor of one topic and can also be the organizer of the seminar. Additionally, there is an explicit dependency which states that the `concluded` attribute from the `Seminar` entity can only be defined after the definition of the `grade` attribute from the `Enrollment` entity. This dependency means that to finish a seminar, the grades for each student enrollment have to be published first.

B. Activity Model and Goal Model

Blended Workflow produces two distinct workflow views based on the same process specification, allowing the user to choose the one that better suits his needs. Whereas the activity model favors process standardization and optimization of work, some people, particularly knowledge workers, may find such restrictions as counter-productive. Therefore, they might prefer the flexible nature of the goal model. The goal model thrives in giving users the power of choice on what to do next, while generally, the activity model only allows a small number of predefined execution paths to achieve the workflow objective.

The activity and goal models differ on the type of work tasks, and the order in which they are executed. One of the main differences is that in the activity model, the sequence in which the instantiation of data elements takes place plays an important role, whereas the goal model is more relaxed in that regard. The activity model may enforce an order of execution in three different ways:

1) The associations between entities require that either both entities are defined at the same time, or one has to precede the other.
2) Explicit dependencies between attributes require that the activity which defines the dependent attribute

should occur after the one which defines the entity that contains the attribute it depends on.
3) It is possible to add additional attribute dependencies in the activity model, but in a way that they do not generate a circular dependency.

On the other hand, in the goal model only 2) applies. In the goal model, entity associations are defined in independent goals. Therefore, it is possible to define two entities and their attributes without requiring them to be associated, because the goal containing the multiplicity condition can be achieved later.

Even though the two models have their differences in terms of the execution flow, both of them share the same ultimate workflow principle: to finish a business process instance, the mandatory entity, along with all its attributes, has to be defined. If that proposition is true, then the business process instance can be finalized, regardless of which model is being used.

C. Seminar Organization Scenario – Activity Model

A possible representation of the management of a university seminar scenario is shown in figure 3. This representation follows a typical BPMN-like activity-based specification enriched with pre and postconditions underneath each activity. The execution starts on the green circle and ends on the red circle. The exclusive gateways denote cycles that allow the creation of multiple instances and represent a condition that has to be true in order to proceed to the next activity.

In the resulting activity model of the seminar organization scenario, we can see that the seminar starts by setting an appropriate title and designating staff members responsible for organizing the seminar. Note that instances of entity `StaffMember`, as well as all its attributes, are assumed to have been previously defined in the system. Taking the "Enroll student in a seminar and assign him a topic" activity as an example, we can see that this operation is represented by the definition of the `Enrollment` entity (as well as some of its attributes) and the association between this entity with `Seminar` and `Topic`, which is represented in the postconditions. Additionally, to execute the activity, those entities must have been defined first, which is represented in the preconditions. To advance to the next activity, the gateway condition stating that there have to be at least 3 students enrolled must be true, otherwise more students must be enrolled first in order to proceed. The process finishes after executing the "Conclude seminar" activity which, upon the definition of the `grade` attribute for each student in the previous activity, enables the process to define the `concluded` attribute from `Seminar`. Remember that this last attribute definition has an explicit dependency stating that it can only be defined after the grades have been submitted. This dependency is naturally enforced by the activity model.

This is just one possible representation of the scenario [7]. The automatic generation of the activity model would have given a lot more finer-grained activities, but by manipulating the model through the merge and split of activities, one can define an activity model that establishes the intended standard behavior. Regardless of the number of activities, the same entities, attributes, and associations definitions should be performed.

D. Seminar Organization Scenario – Goal Model

Compared to the activity model, the goal model offers more flexibility. Workers are not restricted to a predefined sequence of tasks, hence they can use their knowledge to decide what should be done next. That doesn't mean the execution is completely free; some invariant rules such as dependencies and multiplicities should be respected.

A possible goal model representation of the university seminar scenario is pictured in figure 4. Using this approach, the user has more freedom to choose what to execute. Whereas in the activity model the process must start by executing a specific activity, in the goal model the first task can be "Create a seminar", "Create topic" or "Register new student". Note that these goals don't have a precondition, therefore they are enabled to execute from the start. The user may choose the preferred action based on personal experience or even in the case of unexpected situations. In contrast with the activity model, the order in which goals are performed is not relevant here.

Similarly to the activity model, we assume that the workflow designer modified the automatically generated goal model by merging some goals. Had it not been done, we would have a different goal for each entity, attribute, and association definition.

E. Execution Behavior

Whether it is a goal or an activity, each work task is composed of preconditions and postconditions. A precondition is a requirement that needs to hold to enable the work task execution. This can be a condition stating that to execute the work task, a given entity or set of attributes must have been previously instantiated. A postcondition establishes what is defined within a work task; what work the task produces. This can be the definition of new entity or attribute instances, as well as the creation of associations between entity instances.

As an example, take a look at the final activity of the university seminar scenario's activity model – "Conclude seminar". It comprises the following information:

1) An instance of `Seminar` already exists.
2) An instance of `Enrollment` already exists.
3) An instance of `grade`, belonging to the entity instance `Enrollment` already exists.
4) The association between entity instances `Enrollment` and `Seminar` was previously established.
5) This task defines the attribute instance `concluded`, which belongs to entity instance `Seminar`.

F. Execution Interchange Between Activity and Goal Model

A given process instance can be fully executed according to the activity or goal model, or it may switch between them – hence the "blended" nature of this workflow approach

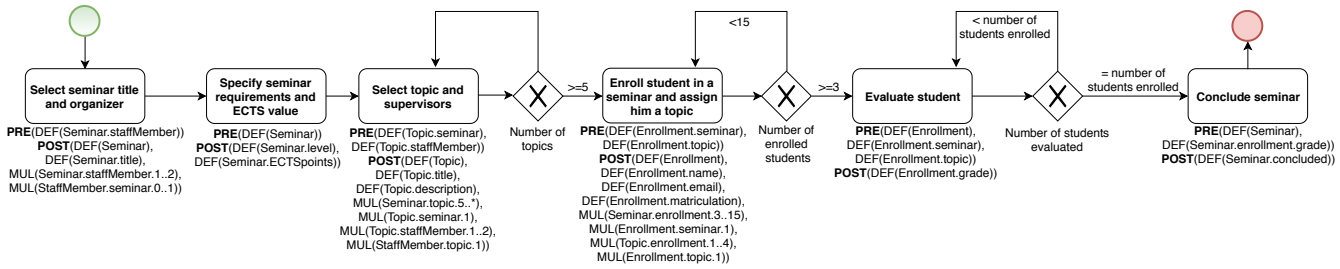


Fig. 3: Activity model of the university seminar scenario.

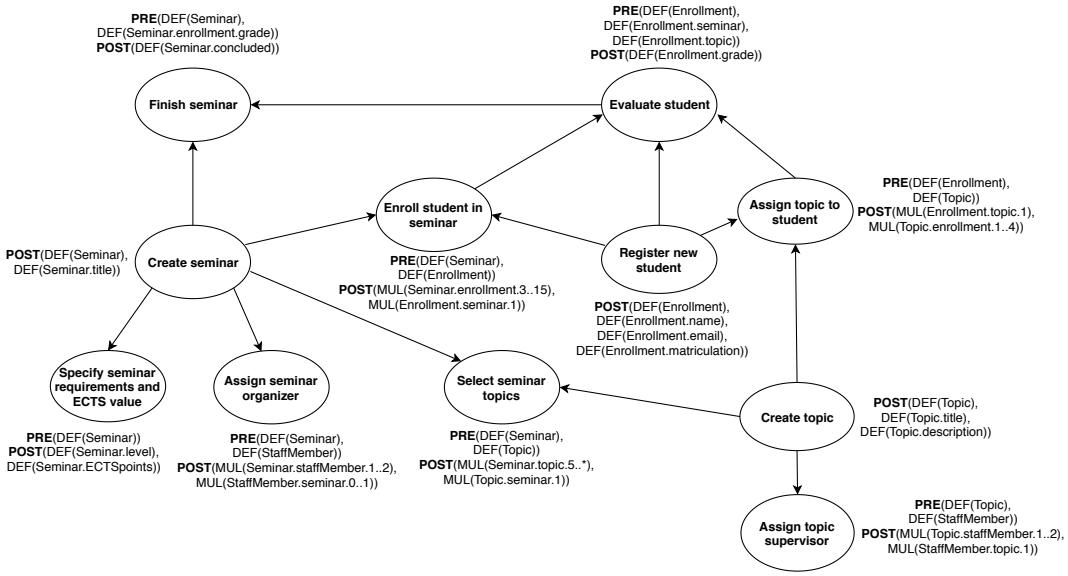


Fig. 4: Goal model of the university seminar scenario.

[5], [6]. Therefore, each view presents the current state of the workflow instance to the end-user, who can switch between them without having to redo any work already done while using the other view. Executing an activity or goal will impact the other model since they are just two different representations of the process' state. Therefore, when the execution switches from one model to the other, some properties should be taken into account.

The end-user can either switch the execution from the activity model to the goal model or do the inverse, from the goal model to the activity model. The most common situation is to switch the execution from the activity model to the goal model, due to an expected situation that the knowledge worker can address. In this case, the execution can proceed in the goal model if its tasks are finer-grained because whatever was already executed in the activity model will correspond to the complete execution of goal tasks. This is what is expected from the goal model, that it can be executed at a lower granularity and in the absence of any unnecessary constraint.

However, it is more complex to switch the execution from the goal model to the activity model, which corresponds to the situation where after having dealt with the unexpected situation it is desirable to return to the standard behavior. This can happen in several cases. Consider the situation where an attribute `A` of entity `E` is set in the goal model (this is only possible if `E` is already defined). The following cases can occur in the activity model:

1) There is an activity enabled for execution (with a valid precondition) whose only postcondition is the definition of `A`. This activity will be automatically executed (the activity is performed without user interaction). The following activities with a valid precondition are then displayed for execution.
2) There is an activity that is not enabled for execution whose only postcondition is the definition of `A`. This activity is also automatically executed, even though it has an invalid precondition. The following activities with a valid precondition are then displayed for execution. Initially, the activity was not available for execution because the data elements in the precondition have not been instantiated yet. After the automatic execution of the activity, those data elements will remain not instantiated, as only the data elements in the postcondition are instantiated. Therefore, if the data elements from the precondition are needed to activate other activities, they must be instantiated in the goal model, or in an activity whose postcondition is the

definition of those data elements (this activity may not be currently available for execution).

3) There is an activity that defines `A` and other attributes. This activity is only partially complete, and Blended Workflow blocks its execution. The user is forced to switch to the goal model and perform the goals that correspond to the execution of the entire activity. In doing so, the activity is automatically executed, since all the work was performed in the goal model.

Unlike in the activity model, when it comes to the definition of associations in the goal model, the entities that are part of the association have to be defined beforehand. Let's suppose we want to define entity `E` and its association with another entity. In the activity model, this is done in a single activity. Upon the definition of `E`, we also specify its association. However, in the goal model, these are two separate goals. If we start by executing the goal that defines `E` and then switch to the activity model, the activity that defines `E` will be blocked since only a part of its postcondition is achieved. To unblock it, the user should switch back to the goal model and perform the goal that establishes the association of `E` with the other entity. After doing so, the activity's postcondition is then fully complete, and the execution may continue in the activity model.

It is important to emphasize that even with these side effects, it is always possible to conclude a process instance. Interchanging the execution among the two models may present a few challenges in some particular situations, but even in the worst-case scenario, the final goal is always achievable. As mentioned, when the execution interchanges between the two models, there can be a point where it is not possible to progress in one of the models because the most recent task is blocked due to being partially executed (a part of its postcondition was achieved using the other model). In this case, the user is forced to switch to the other model and perform tasks that satisfy the entire postcondition of the blocked task. After doing so, the previously blocked task is completed and automatically executed, and the process flow can then continue executing normally.

III. SKIP OPERATION

In a real scenario, it is plausible that the activity model of a given business process consists of a chained sequence of activities. If this is the case, in order to finish the workflow, all activities must be executed so that the final one is reached. This is never the case in the goal model execution. The goal model always allows users to perform only a minimal set of goals that satisfy the essential workflow requirements. In other words, the goal model execution allows to implicitly skip over some non-mandatory work tasks. The main driver of our work is to implement a similar mechanism in the activity model, that allows users to skip unwanted activities, but in an explicit fashion, while still preserving the standard behavior to a certain extent. By doing so, the activity model becomes more versatile and flexible. In terms of semantics, the skip operation allows the execution of the activity model to deviate from its predefined specification, while still being able to achieve the business process' final goal following the standardized sequence.

There is not a definitive way to implement the skip operation in a workflow management system. Regardless of the approach, one problem is always present: if a work task is skipped, then some data will not be provided. To remedy this problem, some suggest providing default values for the missing data, entering the data manually later, or even using established values from previous process instance executions [8]. In our implementation, we tackle this problem in two different ways:

1) Using Blended Workflow's ability to switch the execution between the two models.
2) By applying a mechanism called dependency tree to the activity model.

These solutions will be elaborated further forward.

In this section, we analyze how we implemented the skip operation in a way that is consistent with Blended Workflow's semantics, and how we managed to address some of the problems that skipping data input rises.

A. Internal Management of the Skip Operation

In Blended Workflow, a given activity with a set of work items can only be either fully executed or skipped. It is not possible to execute some work items and leave others to be skipped or executed later.

To understand how the skip operation works, first we need to understand how Blended Workflow internally manages work tasks. At any point in the execution of the activity or goal model, Blended Workflow dynamically discovers the available tasks. Blended Workflow's engine checks all preconditions and postconditions of every task and based on which data elements are already created, recognizes which tasks should be performed next. At the base of this mechanism is data instantiation. We have seen that when a given task executes, the result of the postcondition comes to fruition, and new data instances, entities and attributes, are generated. These entities and attributes serve as the candidates to check whether there is any task whose precondition is now satisfied following the new data elements instantiation.

When an activity is skipped, the same principle is followed. When designing the skip operation, we did not want to "reinvent the wheel"; we did not want to fundamentally change the way Blended Workflow works. As a consequence, we had to come up with a solution that would coexist with the current Blended Workflow behavior. For this reason, we realized that when an activity skips, new data products would still have to be instantiated, since this is the only way to enable new tasks and allow process progression. Therefore, whenever an activity is skipped, the same data products will be instantiated, just like when the activity is ordinarily executed. By doing so, the activity achieves its postcondition and is considered complete, thus not showing available for execution anymore. The process flows naturally, and the next available activities respect the activity model order since the data elements instantiation order was the same as when

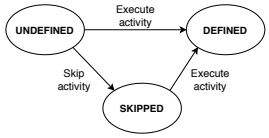


Fig. 5: State transitions of data elements.

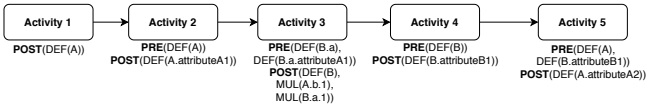


Fig. 6: Activity model with two entities.

activities execute normally. However, these newly skipped data products must be different from the ones created when the activity executes normally, otherwise, there would be no distinction between them. The way we differentiate normally defined product instances from the skipped ones is by using the concept of state.

B. Data Elements State

The notion of state is at the heart of our skip implementation. However, our application of state is not achieved in a conventional way. Some approaches [8], [9] apply the concept of state to the activities themselves. Usually, there is a set of predefined states the activity could be in at any point in time. For instance, an activity is initially in the *Initial* state, when executes it goes to the *Completed* state, and when it is skipped goes to the *Skipped* state.

We do not apply state to activities. Instead, data elements convey the state. One of the reasons for doing this is to be consistent with Blended Workflow's functioning. As we have seen, Blended Workflow enables new activities based on data elements instantiation which in turn fulfill the preconditions of new activities. Therefore, the instantiation of entities and attributes plays a crucial role in process execution, thus we wanted to preserve the act of instantiating data elements. By having data elements in different states, we can tell which ones are well defined and which ones were the target of a skip action.

There are three possible states for each entity and attribute instance: *Undefined*, *Defined*, and *Skipped*. The possible state transitions are demonstrated in figure 5. Every data element is instantiated in the *Undefined* state. If a work task's postcondition contains the data element, then the data element will become either *Defined*, if the work task is executed normally, or *Skipped*, if the work task is skipped. When an entity or attribute reaches the *Defined* state, it cannot be changed anymore; *Defined* is the final state. However, when a data element is in the *Skipped* state, it can still be defined and achieve the *Defined* state. This happens when an entity or attribute instance was initially part of a skipped activity, but later that data instance is redefined. Attribute instances in the *Undefined* and *Skipped* states do not have a particular value, but when they are in the *Defined* state, a value is always provided.

We have seen that when an activity executes, it produces defined instances specified by the activity's postcondition. So, what happens to the activity's postcondition when the activity is skipped? The same data will be instantiated as if it was ordinarily executed, but the instances produced will be in the *Skipped* state.

C. Behavior of the Skip Operation

Earlier we discussed the data model of a Blended Workflow business process and distinguished between mandatory and non-mandatory data. In our skip implementation, we impose that the mandatory data should be always provided because it holds crucial process information. Activities incorporating mandatory data are not allowed to be skipped. This includes activities that define the mandatory entity or any of its attributes. On the other hand, any activity that does not contain mandatory data elements can be skipped. Therefore, in any activity model, there will always be a set of unskippable tasks, considered essential for process completion.

However, activities in which the mandatory data element only appears in the association clause may be skipped. This means that if an activity defines a non-mandatory entity, and the association of that entity with the mandatory entity is part of the activity's postcondition, the activity can be skipped, but the association should be set. The user should select the instances of each entity to be associated and only then skip the activity. By doing so, the mandatory entity becomes associated with a skipped entity instance, that is not yet defined. We enforce this kind of behavior because even when an activity is skipped, the system must know which data instance an entity is associated with, even if it is a skipped entity instance, which is relevant to preserve the sequence of execution defined by the activity model. A similar behavior applies to other skip actions: if an activity is meant to define a non-mandatory attribute, in order to skip that activity, first we need to indicate to which entity instance that attribute belongs to, regardless of whether that entity instance is skipped or defined.

Let's say that in our university seminar scenario we want to skip the "Evaluate student" activity. In other words, we do not want to define a value for the student's grade. Even if we intend to do that, first we need to indicate the student that is the target of the skipping action. There might be several students and we only want to skip the evaluation of one (the student could be sick and therefore would be evaluated later), hence we have to select which student instance we are referring to.

To clarify the behavior of the skip operation, let's look at the activity model pictured in figure 6. The model has two entities: `A` and `B`. Entity `A` is mandatory, and is composed of attributes `attributeA1` and `attributeA2`. Entity `B` is composed of attribute `attributeB1`. Entities `A` and `B` have an association with the cardinality of one-to-one. If the user intends to skip the definitions regarding the non-mandatory entity `B`, the following will occur:

1) Activity 1 is the first activity, and it has to be executed

because in its postcondition there is the definition of the mandatory entity A. The result of Activity 1 is the creation of an instance of entity A in the *Defined* state.

2) After Activity 1 executes, Activity 2 becomes enabled since it now has a valid precondition. The execution of Activity 2 is required to proceed, and cannot be skipped because in its postcondition, there is the definition of the mandatory attribute A.attributeA1. Therefore, Activity 2 creates an attribute instance of attributeA1 in the *Defined* state. The user must select the entity instance this attribute belongs to. In our example, the user may select the entity instance created in the previous activity.
3) The next activity is Activity 3. Activity 3 can be skipped, because in terms of entities and attributes definitions, it only defines the non-mandatory entity B. Hence, if activity 3 is skipped, an entity instance of B will be created with its state set to *Skipped*. Even though the activity can be skipped, the association should be set. Thus, before skipping the activity, the user must select the entity instance of A (it could be the one created in Activity A) as the one that the newly created B will be linked to.
4) Activity 4 is enabled because its precondition is satisfied; an entity instance of B was created in the previous activity. Activity 4 can be skipped since its postcondition only contains B.attributeB1, which is a non-mandatory attribute. However, in order to be skipped, the user must select the entity instance that the attribute belongs to. In our example, it could be the entity instance B created in the previous activity. Whether there is an entity instance in the *Defined* or *Skipped* state, it has to be selected. The result of skipping Activity 4 is the creation of the attribute instance B.attributeB1 with no defined value and in the *Skipped* state, belonging to the entity instance B created in Activity 3.
5) Activity 5 is the final activity, and it cannot be skipped since it has a mandatory attribute in its postcondition. To perform this activity, the user must choose the parent entity instance of this new attribute instance. In our example, it could be the entity instance A, created in Activity 1. Executing Activity 5 will result in the production of the attribute A.attributeA2 in the *Defined* state.

The result of this execution is the creation of the attributes attributeA1 and attributeA2, belonging to entity instance A, all in the *Defined* state and with specified values; and entity instance B with attribute instance attributeB1, all in the *Skipped* state with no values. The entity instances created are associated with each other.

An alternative to the execution steps presented above could be a scenario where the user decides to execute Activity 4 instead of skipping it. Like in step 4), an entity instance of B has to be selected. The selected entity instance could be the entity instance from 3), an instance of B in the *Skipped* state. In this case, since we want to execute the activity, a specific value for attributeB1 has to be provided. After selecting the parent entity instance and providing the attribute value, the activity can be executed. In doing so, an instance of attribute attributeB1 is generated in the *Defined* state and with the specified value. This newly created attribute instance belongs to the entity instance created in step 3), but now, since it has an attribute in the *Defined* state, entity instance B automatically converts itself from the *Skipped* state to the *Defined* state.

This example demonstrates a particular behavior of skipped data elements: when a skipped entity instance is chosen as the parent entity instance in a work task that defines an attribute, if the task executes rather than skips, then the aforementioned entity instance will go from the *Skipped* state to the *Defined* state, as it would not make sense to have a skipped entity instance containing defined attributes. When an entity instance has defined attribute instances, the entity instance itself also has to be in the *Defined* state.

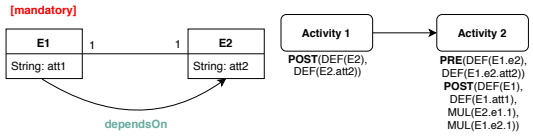
D. Attribute Dependencies

When an attribute definition is skipped, two situations can happen. Typically, that attribute will not be needed in the future, therefore it can stay permanently in the *Skipped* state, with no specified value. However, there might be a case where it is necessary to redefine a skipped attribute instance to proceed with the execution. Such cases occur when attributes have dependencies between them. A given attribute att1 depends on attribute att2 when the information of att2 is required as a prerequisite to define att1. We saw an example of a dependency in the university seminar scenario, where the conclusion of the seminar requires the students' grades first.

Attribute dependencies can be chained. For instance, attribute att1 depends on attribute att2, which in turn depends on attribute att3, and so on. Furthermore, if the multiplicity condition states that multiple entity instances should be associated, we realize how complex the management of all these dependencies can become.

We developed a mechanism called dependency tree whose function is to assist the management of attribute dependencies. One of the dependency tree's goals is to support the awareness and understanding of the dependency chain of a given attribute instance. This is done by providing a visual tool that is particularly helpful when there are several attribute instances involved in the dependency chain. By using the dependency tree, the user can see every dependency of the current attribute instance in a single window. Being aware of every piece of information available helps users to make the best possible decision.

The dependency tree is also used to redefine previously skipped attribute instances within a dependency chain. Going back to the situation where attribute att1 depends on attribute att2, what happens if att2 is skipped? att2 will not have a specific value, therefore the value of att1 cannot be resolved. If the attribute att1 is not mandatory,



(a) Data model. (b) Activity model.

Fig. 7: Models with two entities.

then it can also be skipped. But if `att1` is mandatory, a value must be provided. Besides, even if `att1` is not mandatory, the user might want to define it. The dependency tree solves this problem, by providing a way to reassign values to skipped attribute instances in the dependency chain. In our example, the activity which defines the attribute `att1` will have the option to check the dependency tree, and in doing so, we will see the current attribute instance `att1`, associated with one or many skipped attribute instances of `att2`. At that stage, we can provide new values for the skipped instances of `att2`. If all instances of `att2` have specific values, the attribute instance `att1` can now be defined since all its dependencies are in the *Defined* state. The same behavior applies whether the tree has one depth level (just one dependency in the chain), or if it has several.

Dependency trees support attributes with multiple dependencies. These are attributes that depend on more than one attribute. For instance, the final grade of a student might depend on the student's exam grade, class attendance rate, and homework evaluation.

Consider the data model pictured in figure 7a. It has two entities: `E1`, the mandatory entity, and `E2`. Each entity contains one attribute. Attribute `att1` belongs to entity `E1`, and attribute `att2` belongs to entity `E2`. This data model has the following dependency: `E1.att1 dependsOn E1.e2.att2`, which is an explicit dependency stating that `att1` is dependent on the value of `att2`.

The activity model of this data model is presented in figure 7b. It is composed by two activities: Activity 1 defines entity `E2` and its attribute `att2`. Activity 2 defines the mandatory entity `E1` and its attribute `att1`, as well as the association between the two entities.

Let's suppose that the user executing the workflow decides to skip Activity 1. `E2` and `E2.att2` go to the *Skipped* state, and the attribute `E2.att2` has no value. Activity 2 must be executed since it holds mandatory data. However, since the definition of `E1.att1` requires that `E2.att2` is defined and with a specific value, the user has to utilize the dependency tree to assign specific a value to `E2.att2`. By doing that, both the entity `E2` and the attribute `E2.att2` go to to the *Defined* state. After defining `E2.att2`, the attribute `E1.att1` can now be defined, and the activity's postcondition achieved. Notice that by defining previously skipped data instances, we are performing more actions than what is suggested by Activity 2's postcondition. We are not only achieving the postcondition of Activity 2 but also redefining some extra data elements from Activity 1's postcondition. This always happens when we use the dependency tree to redefine skipped instances, which is why this is considered an exception mechanism; we are performing actions that go beyond the activity's specification.

Since the dependency tree mechanism has the general purpose of aiding the users' visualization and comprehension of attribute dependencies, as well as redefining skipped attribute instances, it is also incorporated in the goal model. In the next section, we will see how the goal model offers another option to redefine such instances.

E. Goal Model Integration

The act of skipping an activity can bring the goal model into play. In the previous section, we saw how it is possible to redefine some skipped data instances by using the dependency tree in the activity model. That can also be achieved by switching the execution to the goal model.

By using the dependency tree we can only redefine the data instances that are part of dependency chains. With the goal model though, it is possible to redefine virtually any skipped data instance. Whether the data is non-mandatory or required because it impacts future decisions, it can be redefined using goals. We saw that data in dependency chains has to be specified to allow the dependent attributes to be filled in. But non-mandatory data may be as important. In the university seminar scenario, think of a situation where we might want to enroll a new student in the seminar, but he doesn't have an assigned institutional email yet. In this case, we want to skip the activity that defines the student's email and proceed with his enrollment. A few days later, he may already have the email, therefore we want to go back and execute the email activity, even though it is not mandatory data. This kind of behavior is only possible in the goal model. The goal model allows achieving process completion, where all data instances are well defined, even if some of them were skipped in the activity model.

We saw that when an activity is skipped, the activity model steps forward the skipped activity, displaying the following available activities. However, in the goal model, when an activity is skipped, the goals that correspond to the skipped data elements are preserved. If these goals are selected, they can serve to either generate new defined data instances or, by selecting the skipped instances, we can redefine them. To maintain process flow consistency, when an activity is skipped, the goal model also moves forward, as if the activity was executed, so that new goals are available as well. As explained previously, when the skipping occurs, new data elements are instantiated and used to enable the preconditions of future tasks. This applies to both activities and goals. Therefore, when an activity is skipped, the goal model preserves the goals that were available to execute prior to the skip action and adds new goals that are now enabled to execute following the skip operation.

To illustrate the collaboration between the activity and goal models, consider the activity model depicted in figure 7b again. Remember that entity `E1` is mandatory. Let's assume the user decides to skip Activity 1 because it only has non-mandatory data elements. In the activity model, Activity

1 will not be displayed again as it was skipped, but the goal model will maintain the goals that achieve Activity 1's postcondition. These goals will be preserved as long as they are not executed in the goal model. Executing these goals will redefine the entity instance `E2`, and the attribute instance `E2.att2` which were instantiated as *Skipped* in Activity 1.

Contrarily to activities, goals are not allowed to be skipped. The goal model is flexible enough as it is and allows users to choose the order in which work tasks are executed. The purpose of implementing the skip operation is to allow flexibility while preserving some of the standard sequence of execution as defined in the activity model.

IV. RELATED WORK

On the first steps of this work, we tried to study and contextualize what the workflow research already had to offer, and in which ways the notion of flexibility was being applied. In particular, we tried to find workflow approaches that have successfully implemented the skip functionality, and understand how they managed to achieve that. Since declarative approaches are usually better prepared to deal with knowledge-intensive and dynamic work environments, we ended up gravitating towards them since the skip mechanism is considered an operation to use in exception cases. However, our goal was to adapt a declarative and flexible mechanism such as the skip operation, to an imperative workflow approach. In particular, our solution allows the activity model to deviate the execution flow from the predefined specification. In this section, we summarize how some workflow approaches apply the skip operation in their implementations, and how it compares to our approach.

One of the most popular methods to deal with change and volatile environments is the case handling approach [1], [9]. It has a data-driven nature that diminishes the importance of activities in favor of the global context, and data dependencies are the primary driver of the process. Case handling conceives the skip operation by applying user roles, where only predetermined users are authorized to skip work tasks. To skip an activity, all preceding activities that have not been completed yet have to be skipped or completed beforehand. An activity may only be skipped if its mandatory data object values are already defined. This is checked by preconditions. In our skip implementation philosophy, we share one of the foundations of case handling's skip: the mandatory data input cannot be skipped. However, in our implementation, the skip mechanism occurs at the granularity of the activity and it is an atomic operation, in the sense that the entire activity is either fully carried out or skipped. Unlike in case handling, we do not allow the definition of some data elements from the activity, namely the mandatory data elements, and then skipping the remainder of the activity. At the same time, similarly to case handling, we also enforce that in order to skip an activity, all previous activities must have been either fully completed or skipped. Though case handling allows redefining skipped data elements by going back to a previous point of execution and redoing a task, it enforces that all following tasks have to be redone as well. In Blended Workflow, it is possible to redefine skipped data elements by switching the execution to the goal model and performing the goals which contain the skipped data elements. The advantage here is that users do not have to redefine all the following data elements afterward; they can just define the desired skipped element. On the other hand, unlike in case handling, we do not allow the redefinition of mandatory data, though this kind of data can never be skipped in either implementation. So, in our approach, the skip is subordinated to the standard sequence of execution, while in case handling it is a mechanism for flexibility, which we can have by using the goal model.

Some approaches only allow skipping tasks if they are marked as such in design time. This means the designer that models the workflow has to explicitly indicate which tasks are allowed to be skipped. This is the case with the Product-Based Workflow Design approach [10], which is a method inspired by manufacturing principles, that takes the product specification and three design criteria as a starting point, after which formal models and techniques are used to derive a favorable new design of the workflow process. In this approach, since a task corresponds to information regarding a single data element, by skipping a task, the information corresponding to that data element will not be provided, meaning the data element is not necessarily required to achieve the process goal. Our proposal differs from this method regarding the phase in which a task is "marked" as skippable. Whereas the Product-Based Workflow Design requires the designer to indicate the skippable tasks at design time, in Blended Workflow they are automatically specified as skippable in execution time. During runtime, based on the nature of the work items a task has, Blended Workflow checks if the currently available tasks are in valid conditions to be skipped or not. If the task contains mandatory elements, then it cannot be skipped. In terms of the structure of work tasks, unlike in Product-Based Workflow Design, Blended Workflow's tasks can be composed of several items and data elements. Therefore, whenever a task is skipped, all the elements within the task will not have an assigned value, contrasting with the Product-Based Workflow Design, where skipping a task only impacts at most one data element. In terms of skipping behavior, we believe our implementation has the upper hand when compared to Product-Based Workflow Design's. By checking if a task is skippable at runtime, there is no need to mark it as such in design time. This grants more user choice since all skippable tasks will be able to be skipped by default. Additionally, Product-Based Workflow Design's skip implementation is prone to design errors. If the designer forgets to mark a task as skippable, and later a user intends to skip it, the specification model has to be changed to represent that. This is never the case in Blended Workflow; it is always possible to skip a task as long as it has valid conditions.

The work of [2] elaborates on the notion of predefining the skippable tasks in the process model. Then, during execution, only those tasks can be skipped. There is a suggestion that the skipped tasks could be executed at a later stage. In Blended

Workflow, the skipped tasks will not reappear for execution in the future. That doesn't mean the skipped data elements will be permanently in the *Skipped* state. By exchanging the execution to the goal model, users can execute goals that redefine skipped data elements. It is important to highlight that this behavior is not the same as redoing the skipped task. An activity is initially skipped, and later a goal redefines the data elements from the skipped activity. We have discussed previously the advantages our skip implementation has when compared to approaches that enforce the marking of skippable tasks at design time. Additionally, in Blended Workflow, there is no need to execute skipped activities later to fill in missed data. Skipped data may be defined by executing the corresponding goals in the goal model, or by necessity if another activity requires them as preconditions.

In the approach suggested in [8], workflow activities can have different states, including the *Skipped* state. According to this proposal, activities can only be skipped when they are in the *Initial* state, meaning the activity cannot be currently executing in order to be skipped. When an activity is in the *Initial* state and the user decides to skip it, the activity goes to the *Skipped* state. This approach also stresses the consequences that skipping an activity can have on subsequent activities, namely the fact that the data of the skipped activity is not provided and future activities might require that information to be carried out. It is mentioned that to cope with these issues, whenever an activity is skippable, the way to provide the data needed in the remainder of the process has to be defined by the business process model. Possible solutions include entering the data manually afterward, using established values from previous process instance executions, or providing default values for the missing data. In our Blended Workflow skip implementation, activities do not have state. Instead, the notion of state is confined to data elements like entities and attributes. When an activity is skipped, the data elements it contains go to the *Skipped* state. However, they do not stay in that state permanently. If the data element is needed later due to a dependency, or because the user did not want to perform the activity at the time, it is possible to redefine the data element by executing the correspondent goal in the goal model, or by using the dependency tree in the case of the activity model. By doing so, the data element goes to the *Defined* state, and in the case of an attribute, a specific value is assigned. In their approach, it is necessary to redefine the workflow semantics to cope with the skipped data that is actually needed.

As we have seen, our proposal differs from others in various ways. The use of user roles and the need to declare a task as skippable in design time are examples of areas where our implementation diverges from the ones studied. The Blended Workflow's coexistence of two different models at execution time makes our implementation of the skip operation specific to this workflow management system. Only tasks in the activity model are allowed to be skipped, but by switching the execution to the goal model, it is possible to solve the problems that the potential missing data could provoke. Due to its declarative properties, the goal model does not require the skip operation.

Our work is distinguishable in the sense that we enable the skip operation in an imperative workflow approach (the activity model), whereas the approaches studied do it on purely declarative workflow methods, where flexible operations are allowed by default. In our implementation, with the help of the context of a declarative approach, users can execute a standardized workflow specification, and still deviate the execution from the predefined specification. This kind of behavior is rarely seen in imperative approaches.

V. CONCLUSIONS

Nowadays, an effective workflow management system is at the heart of productivity in any corporate workplace. Flexibility is an important workflow characteristic in knowledge-intensive environments, where exceptions may occur frequently. Therefore, a workflow capable of adapting to the situation in hand is pivotal to success.

In this work, we provide a way to increase the level of flexibility of an imperative workflow approach. In particular, we explain how we implemented the ability to skip activities on the Blended Workflow tool while maintaining process consistency. Skipping information input comes at a cost, but by taking advantage of the dependency tree, and Blended Workflow's declarative model, we can mitigate those issues.

Lastly, in section IV, we take a look at other credible approaches, explain how they achieve the skip operation, and compare their implementations to our own.

The code of Blended Workflow, including the skip operation, is publicly available in a GitHub repository[2].

REFERENCES

[1] C. W. Günther, M. Reichert, and W. M. Van Der Aalst, "Supporting flexible processes with Adaptive Workflow and Case Handling," *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE*, pp. 229–234, 2008.

[2] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W. Van Der Aalst, "Process flexibility: A survey of contemporary approaches," *Lecture Notes in Business Information Processing*, vol. 10 LNBIP, pp. 16–30, 2008.

[3] N. Mulyar, M. Pesic, and M. Peleg, "M.: Towards the Flexibility in Clinical Guideline Modelling Languages," *Information Systems Journal*, pp. 1–18, 2007.

[4] M. Reichert and B. Weber, "Enabling flexibility in process-aware information systems: Challenges, methods, technologies," no. October, pp. 1–515, 2012.

[5] A. R. Silva, "A formal verification of the integration of activity and goal-based workflows," submitted for publication.

[6] A. R. Silva and V. García-Díaz, "Integrating activity- and goal-based workflows: A data model based design method," in *Business Process Management Workshops*, M. Reichert and H. A. Reijers, Eds. Cham: Springer International Publishing, 2016, pp. 352–363.

[7] M. Hewelt and M. Weske, "A Hybrid Approach for Flexible Case Modeling and Execution," *Lecture Notes in Business Information Processing*, vol. 168, no. October 2017, pp. 1–23, 2013.

[8] M. Weske and S. Straße, "Flexible Modeling and Execution of Workflow Activities," *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, vol. 7, no. c, pp. 713–722 vol.7, 1998.

[9] W. M. Van Der Aalst, M. Weske, and D. Grünbauer, "Case handling: A new paradigm for business process support," *Data and Knowledge Engineering*, vol. 53, no. 2, pp. 129–162, 2005.

[10] H. A. Reijers, S. Limam, and W. M. P. van der Aalst, "Product-Based Workflow Design," *Journal of Management Information Systems*, vol. 20, no. 1, pp. 229–262, 2003.

[2] https://github.com/socialsoftware/blended-workflow/tree/SkipRedoExecution