Blended Workflow: Introduction of the Skip Operation

André Pereira Rodrigues

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. António Manuel Ferreira Rito da Silva

Examination Committee

Chairperson: Prof. Luís Manuel Antunes Veiga
Supervisor: Prof. António Manuel Ferreira Rito da Silva
Member of the Committee: Prof. Maria Dulce Pedroso Domingos

November 2019

Acknowledgments

I would like to thank my family and girlfriend for all the support given throughout this thesis. Even on the toughest days, I always knew they would be there for me to cheer me up. You are part of my daily life; you know how hard this was for me, and how important every one of you was in this journey.

I would like to express my gratitude for all my friends and colleagues who embarked with me on this step of my life. Finishing this academic course was hard, but you definitely made it a lot easier. I learned a lot with my university friends with whom I made a lot of sacrifices to achieve my academic goals. All the good and bad moments shared with you were precious.

I want to thank all the teachers with whom I had the chance to learn from, especially Professor António Rito Silva, my thesis supervisor, for sharing his knowledge with me and guiding me to be a better student and future professional. This thesis pushed me to learn new things, but most importantly, it made me develop as a person.

Lastly, I want to thank everyone that I'm (definitely) forgetting for being part of my life and helping me be the person I am today.

This thesis wouldn't be possible without all of you.

Abstract

Nowadays, business process management is supported by workflow tools that provide maximum efficiency in resource and personnel administration in order to create the best possible conditions to achieve the business process goals. A flexible workflow is a workflow capable of handling exception cases and adapting the execution to unforeseen situations. Whereas classical workflow approaches restrict the execution to a specified path, flexible workflows allow the execution to deviate from its initial specification. This thesis explains how we implemented a flexible operation – the skip operation – in a particular workflow environment called Blended Workflow. Blended Workflow is a business process management tool that provides two different process views to the user: a traditional imperative model, and a looser declarative model. The skip operation applies to the imperative model, which is a stricter approach by nature. The ability to skip over undesired work tasks allows the execution to deviate from its predetermined specification. The skipped data can be later defined using the declarative view of the workflow. With this new functionality, 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.

At the heart of the skip operation, is the notion of data elements state: skipped data instances have a different state from normally defined data instances. A mechanism called dependency tree is used to manage information dependencies, which is particularly helpful in situations where some data depends on other skipped data.

Keywords

Business Process Management; Workflow; Flexibility; Skip; Dependency Tree.

Resumo

Hoje em dia, a gestão de processos de negócio é auxiliada por ferramentas de *workflow* que proporcionam eficiência máxima em termos de administração de recursos e pessoas, de forma a criar as melhores condições possíveis que levem a alcançar os objetivos do processo de negócio. Um *workflow* flexível é um *workflow* capaz de tratar de casos de exceção e adaptar a sua execução a situações inesperadas. Enquanto que as abordagens clássicas de *workflows* restringem a execução a um caminho específico, os *workflows* flexíveis permitem que a execução se desvie da especificação inicial. Esta tese explica como implementámos uma operação flexível – a operação *skip* – num ambiente de *workflow* particular chamado Blended Workflow. O Blended Workflow é uma ferramenta de gestão de processos de negócio que fornece duas perspetivas diferentes ao utilizador: um modelo tradicionalmente imperativo, e um modelo declarativo mais fluido. A operação *skip* é aplicada ao modelo imperativo que, por natureza, é uma abordagem mais rígida. A capacidade de passar à frente tarefas de trabalho não desejadas permite que a execução fuja à especificação inicial. Os dados que não foram preenchidos podem ser definidos mais tarde usando a perspetiva declarativa do *workflow*. Com esta nova funcionalidade, os utilizadores não têm necessariamente de executar tudo o que está inicialmente especificado pelo *workflow* imperativo, sendo ainda assim possível alcançar o objetivo final do processo de negócios.

No cerne da operação *skip*, está a noção de estado dos dados: as instâncias de dados que foram passadas à frente têm um estado diferente das instâncias de dados preenchidas. A árvore de dependências é um mecanismo usado para gerir dependências de informação, que é particularmente útil em situações onde temos dados que dependem de outros dados que não foram preenchidos.

Palavras Chave

Gestão de Processos de Negócio; *Workflow*; Flexibilidade; *Skip*; Árvore de Dependências.

Contents

List of Figures

Listings

Acronyms

API	Application Programming Interface
BPMN	Business Process Model and Notation
CMMN	Case Management Model and Notation
CSS	Cascading Style Sheets
DTO	Data Transfer Objects
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MVC	Model-View-Controller
PAIS	Process-Aware Information System
PBWD	Product-Based Workflow Design
REST	Representational State Transfer

1

Introduction

Contents

1.1 Context

A good business process management strategy is at the heart of a company's effectiveness, productivity, and commercial success. Whether it is a new project or an ongoing business practice, the way work is performed is key to achieve an enterprise's goals. It is, therefore, logical for any company to try to maximize the time, effort, energy, and resources it takes to accomplish such tasks. In modern-day technology, this translates into having an efficient workflow management tool that can support all business processes within a company.

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 [2]. 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 process types. 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 [3].

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 vary depending on different factors. A common example of this approach is the healthcare industry, where the treatment of patients depends on each case [4].

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 [5]. 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 [1, 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.

1.2 Motivation and Objectives

In this thesis, 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. In the process, we also improve the way dependencies between data elements are perceived and managed by the users, providing them a new user-friendly method to visualize those dependencies.

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 still achievable.

Our intent is, therefore, to provide flexible capabilities that allow a naturally rigid imperative workflow to deviate its execution from the initial specification. In particular, we apply the skip operation to Blended Workflow's imperative model, which in turn, allows the execution to not explicitly follow the predefined path, while still achieving the main workflow goals.

By skipping data input, some information ends up ultimately missing. In this work, we also propose solutions to mitigate these issues, ensuring that even when work tasks are skipped, process consistency is maintained.

1.3 Document Organization

The remainder of this thesis is organized as follows:

- Chapter 2: Workflow Contextualization, presents the main concepts which serve as the foundation of our work.
- Chapter 3: Blended Workflow, explains the overall behavior of Blended Workflow.
- Chapter 4: Implementation of the Skip Operation, describes the implementation process of the skip operation and explains how the solution is coherent with Blended Workflow's current behavior.
- Chapter 5: Comparative Analysis with Other Approaches, overviews other approaches to the skip mechanism, and compares them to ours.
- Chapter 6: Conclusion, presents the closing remarks and proposes some future improvements to our work.

2

Workflow Contextualization

Contents

2.1 Introduction

Technology has changed the way people interact, behave, and work. Our society is constantly evolving through innovative technological ideas that change the way we approach different aspects of life. When it comes to work management, the migration from paper records to computer automation has made a huge impact on the productivity of companies in different working fields. Whether it's educational, medical, sports, entertainment, or any other domain, nowadays workers use computer assistance to support their tasks and increase efficiency. This is where the concept of workflow comes in. A workflow is a tool used to monitor, assist, and guide workers in their work tasks. From a more abstract or higher-level perspective, a workflow may be considered a view or representation of real work. The concept of workflow is heavily linked with business process management and a lot of companies invest in digital workflows to better manage data, resources, and work progression in big projects.

Within a workflow, the business process is broken down into small and quickly manageable tasks. These tasks are then assigned to different workers who use the workflow for guidance. After performing a given task, new tasks become available for execution. The workflow specification dictates the order in which work tasks should be carried out.

2.2 Process-Aware Information Systems

Business process management models describe business processes at a high level of abstraction, serving as a basis for analysis, simulation, and visualization of work. A business process model comprises the process activities and their attributes, as well as the control and data flow between activities [7]. The implementation lifecycle of a traditional workflow, also known as Process-Aware Information System (PAIS), usually starts when the process designer (someone who has a deep understanding of the overall process) creates a general process model. That process model serves as a template for the execution of future process instances. Once a process instance is running, workers can start handling work items. The system might be able to monitor and analyze the execution logs, and based on that information, the process designer can decide to evolve the process to an updated version.

Any PAIS should be capable of the following [7]:

- Explicitly describe processes in terms of executable process models.
- Ensure that work tasks are executed in the specified order or comply with particular constraints.
- Control the flow of data between work tasks; that is, the output data of a particular work task can be consumed as input data by subsequent work tasks.
- Assign work items to the corresponding workers and manage user authorization.

Adaptive processes share a common philosophy with agile software development methods, which focus on encouraging human developers to evolve software rapidly and effectively. Processes are adaptive in the sense that they are continually evolving and reshaping to fit the situation in hand. A PAIS needs to be agile and flexible in order to deal with exceptions, efficiently deal with uncertainty, change the execution of single business cases on the fly, cope with variability, and support the evolution of implemented business processes over time [7]. A PAIS should not freeze the implementation of business processes but instead, allow authorized users to flexibly deviate from the pre-specified process whenever required as well as to evolve process implementations over time. In an adaptive PAIS, the process engineering phase is not performed just at the start, it's done throughout the whole process lifecycle, whenever the need to change or evolve the implementation is made clear.

When it comes to changes, process adaptations can be accomplished at two different levels: the process type level, by providing process model evolution and instance migration, and the process instance level, which might require ad-hoc adaptations.

To have a robust system, errors should be avoided at build time. However, the PAIS should enable users to effectively deal with both expected and unexpected events during runtime. A PAIS can have different exception handling mechanisms for different perspectives such as the process designer, the system itself, the end-user, and the system supervisor [7].

2.3 Differences Between Imperative and Declarative Approaches

There are two main philosophies in terms of workflow implementation: imperative and declarative. They both share some common aspects, such as the division of the whole process into smaller and more manageable tasks, and the target of achieving the successful business process goals. However, each one is more suited to certain situations. Up next, we will discuss some of the areas where the two approaches differ:

- **Process Flow**

  In terms of process sequence, imperative methodologies tend to follow a control-flow approach. This means that new work tasks are enabled depending on what has been done previously; if work task A has been performed, then work task B should be performed next. On the other hand, in declarative approaches the activation of new work tasks depends on which data is currently available; e.g., when a specific email is received, execute the next task.

- **User Interaction**

  Imperative workflow implementations are designed to guide workers in their execution. It is not up to the workers to decide what should be done next; the workflow dictates that. In contrast, declar-

ative workflows are meant to assist workers, rather than guiding them. The users of declarative workflows should be workers who have a deep understanding of the whole process, and they usually prefer to guide the process based on their experience and know-how. A declarative approach focuses on what should be done instead of how.

- **Process Guidance**

  Standardization is the way to go with imperative approaches. This means that all process instances should perform the same work tasks in the same explicit order. Declarative implementations guide each process instance in a case-by-case fashion, where two different process instances may vary from each other.

- **Workflow Nature**

  Imperative workflows tend to be more rigid, whereas declarative workflows are usually more flexible. In other words, unlike in imperative approaches, declarative implementations allow the process execution to deviate from its predefined specification.

- **Work Environment**

  Imperative workflow approaches favor static environments where there is not a great deal of change and variability. Declarative approaches, on the other hand, thrive in dynamic knowledge-intensive environments where change is welcomed.

- **Notation Standard**

  Business Process Model and Notation (BPMN) is the renowned standard for imperative approaches. The standard notation for declarative workflows is Case Management Model and Notation (CMMN).

2.4 Workflow Flexibility

When it comes to the world of workflow management systems, one of the hottest topics is flexibility. Flexibility is key in dynamic workflow environments because it reflects the ability of an organization's processes and supporting technologies to adapt to both foreseen and unforeseen changes. Flexibility is effectively a balance between change and stability that ensures the identity of the process is retained. As we have seen in the previous section, declarative workflow approaches tend to favor flexibility almost by default. Generally, to increase flexibility in an imperative process, more execution paths have to be modeled explicitly, whereas increasing flexibility in declarative processes is accomplished by reducing the number of constraints or weakening the existing ones. The more constraints are defined for a process, the fewer execution paths are possible, thus limiting process flexibility.

2.4.1 Flexibility Taxonomy

In recent years, the broad notion of flexibility in workflow systems has been tackled by different people, using different approaches so that it can be broken down into more specific ideas. An extended taxonomy for flexibility has been proposed [3]. It breaks down flexibility into four different types:

- **Flexibility by Design:** The ability to incorporate alternative execution paths within a process model at design time, allowing the selection of the most appropriate execution path to be made at runtime for each process instance.
- **Flexibility by Deviation:** Ability for a process instance to deviate, at runtime, from the execution path prescribed by the original process without altering its process model.
- **Flexibility by Underspecification:** When all execution paths cannot be defined in advance, it is useful to be able to execute an incomplete process model and dynamically add process fragments expressing missing scenarios to it.
- **Flexibility by Change:** Ability to modify a process model at runtime such that one or all of the currently executing process instances are migrated to a new process model. The work of [8] distinguishes the target of change: it can be either the business process type (a template of how an instance of the process should execute) or the business process instance (a concrete process executing). Whereas type evolution often reflects the redesign of processes, instance evolution is necessary to cope with exceptional situations.

There are different angles when it comes to the way we look at changes [2]. On one hand, we have exceptional change, which contemplates rare and unforeseen events during process execution. Exceptional change requires the process to be ”bent” in order to fit the case in question. On the other hand, we have evolutionary change, which is a planned migration of a process to an updated schema version.

2.4.2 Flexibility Requirements

Some knowledge-intensive processes require flexibility since they are neither fully predictable nor repetitive and therefore cannot be entirely pre-specified at build time. Even in the case of repetitive business processes, which are usually predictable, a certain degree of flexibility may be necessary to support dynamic process adaptation in the case of exceptions. Many business processes can be characterized as a combination of predictable and unpredictable elements. However, knowledge-intensive fields such as engineering, science, or healthcare, often have processes that depend on the context and are situation-specific. This kind of process requires four essential flexibility needs [5]:

- **Variability:** Processes are handled differently depending on the context. Process variants might exist due to differences in regulations found in different countries, groups of customers (premium vs. standard customers), temporal differences, among others.
- **Looseness:** In some processes, only the end goal is known before the execution of the process. The parameters determining the exact course of action are typically not known *a priori* and might change during process execution. As a consequence, these processes cannot be fully pre-specified; they require a loose specification.
- **Adaptation:** Represents the ability of a PAIS to adapt the process and its structure to emerging events. Adaptations might require coping with special situations during the process execution, which have not been foreseen in the process model. Usually, many exceptions can be anticipated and therefore planned upfront by capturing them in the process model. However, it is hardly possible to foresee all exceptions that may occur in the context of a particular process. Therefore, support for dealing with unplanned exceptions is additionally needed. Mechanisms of skipping and redoing activities are examples of this.
- **Evolution:** Ability of the process to change when the corresponding business process evolves. The evolution of real-world processes can be triggered by a changing business context, technological advances, regulatory adaptation, or organizational learning. Evolution can also happen as a result of changes inside the PAIS itself due to design errors, technical problems, or poor internal quality. Process evolution may be incremental or revolutionary. The evolution of the PAIS may impact (or not) the currently running processes. The evolution can be deferred when the old version of the process instances are allowed to complete, but if the evolutionary change affects ongoing process instances, then the evolution should be immediate. Evolutionary changes can be permanent or temporary and may impact the observable process behavior or the internal structure of the PAIS.

To meet these four needs, the PAIS should be able to fulfill some requirements. To achieve variability, the PAIS should allow process configuration. Loosely-specified processes grant looseness. Exception handling and ad-hoc changes are mechanisms used to accomplish adaptation. Evolution is achieved by having versioning, refactoring, monitoring, and process instance migration. To have a truly flexible environment, the PAIS should also take into account features like accountability and traceability (logging), business compliance, access control, correctness of changes, user support, among other qualities.

Within a company structure, changes can have different subjects: the activities themselves, the information output, the authorization access, or the sequence of actions. Those changes can be classified according to different properties such as the extent of the change (incremental or revolutionary), duration (temporary or permanent), swiftness (immediate or deferred), and anticipation (ad hoc or planned).

In a business enterprise context, several factors should be taken into account upon the design and implementation of business process models. The company's size and scale, the culture within the enterprise, the type of industry field it belongs to, or the number of available resources, exemplify some of these differentiating factors [9]. Upon analyzing these factors and identifying the corporation's context, a suitable business process management strategy should be implemented. If the enterprise's working field includes dynamic environments, such as the medical field, then the company should adopt more flexible workflow tools when compared to more static environments, like industrial environments, where there is less probability of occurring unexpected situations.

In some cases where a workflow is not flexible enough, the work that has actually been done might not be precisely represented in the workflow itself because an expected situation occurred, and the rigid workflow was not prepared to deal with it. In this case, workers might try to find alternative solutions. However, by acknowledging the workarounds instead of ignoring them, organizations can perform corrective actions and improve their work systems.

3

Blended Workflow

Contents

3.1 Introduction

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 3.1. On a high-level view, 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 in a way that 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 a final state. The designer can apply a set of manual transformations by reordering, merging, splitting, or renaming work tasks, to enforce a particular behavior in the execution of the business process. By modeling and defining the process and its work tasks, a template model of the business process is generated, and new process instances can be created from this template. It is at the execution phase of such process instances that Blended Workflow shines. Upon execution, Blended Workflow provides two interchangeable options to perform work: the activity model, which is a classic imperative approach; and a more flexible way in the form of the goal model. Both of these models allow users to perform the same amount of work, but they differ in the way they do it. In general, the activity model offers a more strict and straightforward way to perform work tasks, whereas the goal model is more fluid in terms of the order in which those tasks are executed. The user can decide which execution model is preferable, knowing that at any time the execution flow can switch between one model to the other, and the work already done will not be compromised. It is, therefore, up to personal preference or case-specific conditions the manner in which a specific process instance is handled. Whether only one of the models is used throughout the whole process or the execution exchanges between both models, it is assured that the final goal of the business process is always achievable.

Another feature of Blended Workflow is the execution log, which is a complementary tool to process execution that allows users to check which work tasks were already performed, in which order, and by whom. This sort of information can be helpful to understand the history of a specific business process.

Blended Workflow supports the storage and management of multiple business process specifications and instances. Unwanted process instances can also be deleted.

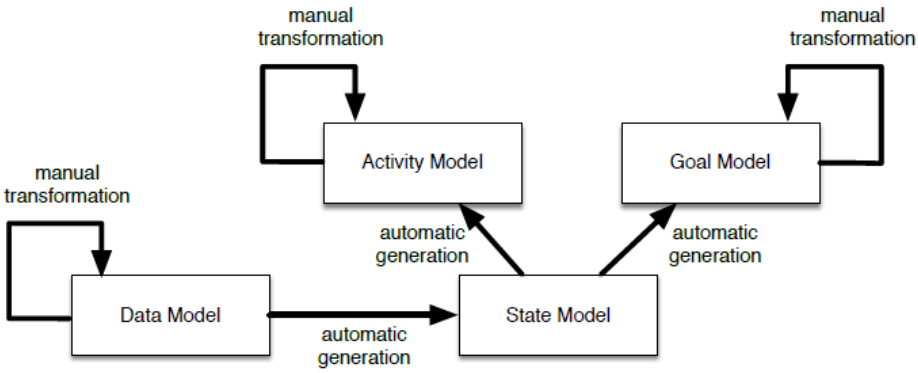


Figure 3.1: Blended Workflow's design process. Image taken from [1].

In the following sections, we will describe in more detail each of the main stages of the Blended Workflow process lifecycle. We will also 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.

3.2 Process Formulation and Design

The first step of the Blended Workflow business process implementation is to define the business process data model. This procedure has to be done by someone who has a deep understanding of the whole business process, is aware of its goals, and can decompose the process in finer-grained tasks. This person should also be able to identify the business process objects and actors, as well as understanding the relationship between them.

Blended Workflow provides a modeling specification tool that allows the designer to define the business process data model. This tool runs on top of the Eclipse Integrated Development Environment (IDE) and uses a particular formal syntax to specify the data model of the process.

Blended Workflow's 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. Drawing a parallel with the

[1] Example inspired by [10].

concepts from object-oriented programming, entities can be seen as objects, a kind of named container that has different fields, which in Blended Workflow's case are the attributes. Therefore, an entity has a unique name and is composed of a set of attributes. Each attribute has a specific data type such as String, Number, or Boolean. Additionally, like in object-oriented approaches, the data specifications serve only as a template for instantiation during runtime. This means that throughout the actual process execution, new entity instances and attribute instances are being generated following their data model description.

Entities are associated with other entities, and those associations have a multiplicity level that informs how many entities a given entity should be linked to.

Every data model has one mandatory entity. This entity, along with all its attributes, must be defined in order to complete a business process instance. In other words, all work tasks that establish values for the mandatory attributes must be performed to conclude the process.

Attributes can depend on other attributes. Such dependencies are explicitly specified in the data model and serve the purpose of enforcing the order of attributes definitions in cases where the information of one attribute requires the information from another attribute first.

3.2.1 Seminar Organization Scenario – Data Model

The data model of the organization of a university seminar scenario is pictured in figure 3.2. 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. For the sake of simplicity, domain objects such as student presentations, papers, or software artifacts are not considered in the data model.

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. Such dependency is formally specified as: `Seminar.concluded dependsOn Seminar.enrollment.grade`. Informally, this dependency means that to finish a seminar, the grades for each student enrollment have to be published first.

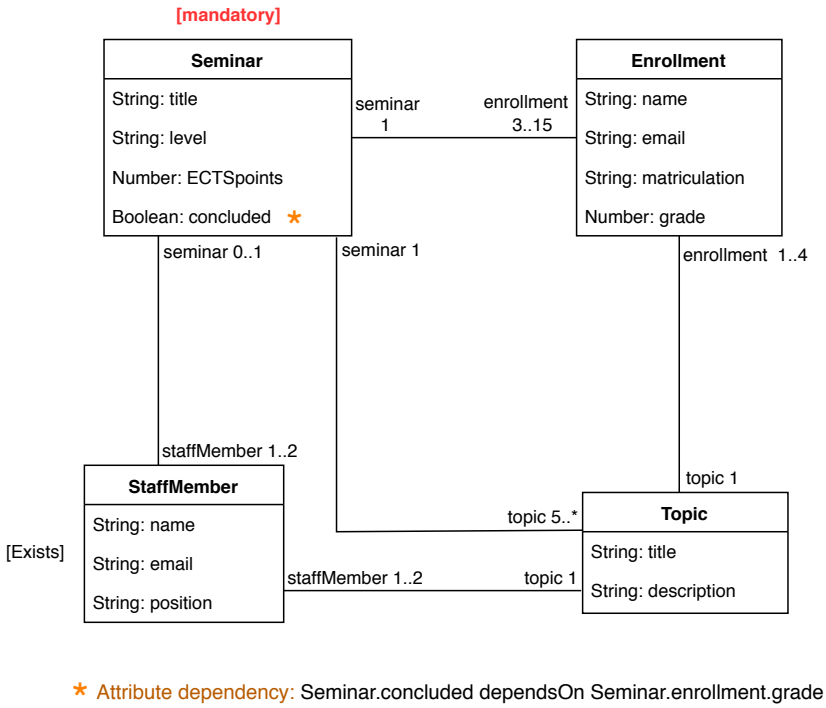


Figure 3.2: Data model of the seminar organization scenario.

Listing 3.1 exemplifies how the university seminar scenario data model is represented using Blended Workflow's modeling tool. It has a particular syntax that all data specifications must follow. In line 1, the business process specification name must be provided. In this case, the specification is called "Seminar". Then, all entities must be characterized along with their respective attributes and data types. As mentioned, the entity `Seminar` has the `mandatory` property, and the entity `StaffMember` is annotated with `exists`, which means that it is assumed that instances of `StaffMember` are already created in advance. Notice the explicit dependency between the attribute `Seminar.concluded` and `Enrollment.grade`. Lastly, we can see how the entity associations and the respective multiplicities are set.

Listing 3.1: Blended Workflow's data model representation of the seminar organization scenario.

```
Specification: "Seminar"

Entity Seminar mandatory {
    title : String
    level : String
    ECTSpoints : Number
    concluded : Boolean dependsOn Seminar.enrollment.grade
}

Entity StaffMember exists {
    name : String
    email : String
    position : String
}

Entity Enrollment {
    name : String
    email : String
    matriculation : String
    grade : Number
}

Entity Topic {
    title : String
    description : String
}

Association SeminarHasStaffMember {
    Seminar with seminar (0..1)
    StaffMember with staffmember (1..2)
}

Association SeminarHasEnrollment {
    Seminar with seminar (1)
    Enrollment with enrollment (3..15)
}

Association SeminarHasTopic {
    Seminar with seminar (1)
    Topic with topic (5..*)
}

Association StaffMemberHasTopic {
    StaffMember with staffmember (1..2)
    Topic with topic (1)
}

Association EnrollmentHasTopic {
    Enrollment with enrollment (1..4)
    Topic with topic (1)
}
```

3.3 Workflow Generation

Based on the provided data model specification, Blended Workflow is then able to automatically generate the activity model and the goal model for the business process. These generated models are effectively two distinct workflow views of the same process. They represent different perspectives of the work to be done, organized in a different order, and following a few different rules. But ultimately, both models allow workers to perform the same amount of work and achieve the business process targets. It is just the way they behave that varies.

Even though the generation of both the activity and goal models is automatically done by Blended Workflow's engine, the designer can then tweak and modify them manually by merging or splitting work

tasks. The system will guarantee that those changes will not introduce any inconsistencies.

Depending on how the specification was modeled by the designer, the number of definitions of each work task can vary from a single entity, attribute, or association definition, to multiple definitions of different natures. It is worth noting that when Blended Workflow automatically generates both the goal and activity models, a single definition is assigned to each work task. This means that if a more complex work task containing multiple attribute definitions is desired, it is up to the designer to model the work task accordingly.

It is also the responsibility of the process designer to name each work task appropriately. Blended Workflow assigns a basic name to each work task, representing the definition that is performed within the task.

Since the goal model is, by default, more liberal than the activity model regarding the sequence order of execution, it will generally have multiple starting points in terms of the opening work task. Thus, when it comes to the goal model, Blended Workflow will automatically generate a workflow implementation that has various starting points, allowing the user to choose different work tasks as the workflow instance's beginning. If a given entity or attribute definition has no prerequisite, then the corresponding work task will be enabled to execute from the start.

However, this is not the case in the activity model. In the activity model, both the order of the work tasks and the user's choice is more restricted in terms of choosing the next work task to perform. Usually, the user will only be able to choose between a couple of work tasks to execute next. The order in which Blended Workflow automatically generates the work tasks in the activity model is randomized, meaning that generally there is no logical explanation as to why the work tasks were generated in a given order. Nevertheless, the process designer can always model the automatically generated activity model to enforce a specific order.

3.4 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.

Each activity or goal represents a work task that defines a set of entities, attributes, or associations. Though effectively they are not necessarily different, work tasks are called activities in the activity model, and goals in the goal model.

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.

In general, the order in which two process instances execute following the goal model can be completely different, contrasting with the activity model, where two distinct process instances will likely have very few differences in terms of the execution flow order. Even though the activity model tends to approximate all process instances execution to one uniform execution flow, parallel flows are allowed, meaning that there might be some variations between the process instances.

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.

3.4.1 Seminar Organization Scenario – Activity Model

A possible representation of the management of a university seminar scenario is shown in figure 3.3. 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. Dotted arrows represent the entities' usage in terms of preconditions and postconditions. 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.

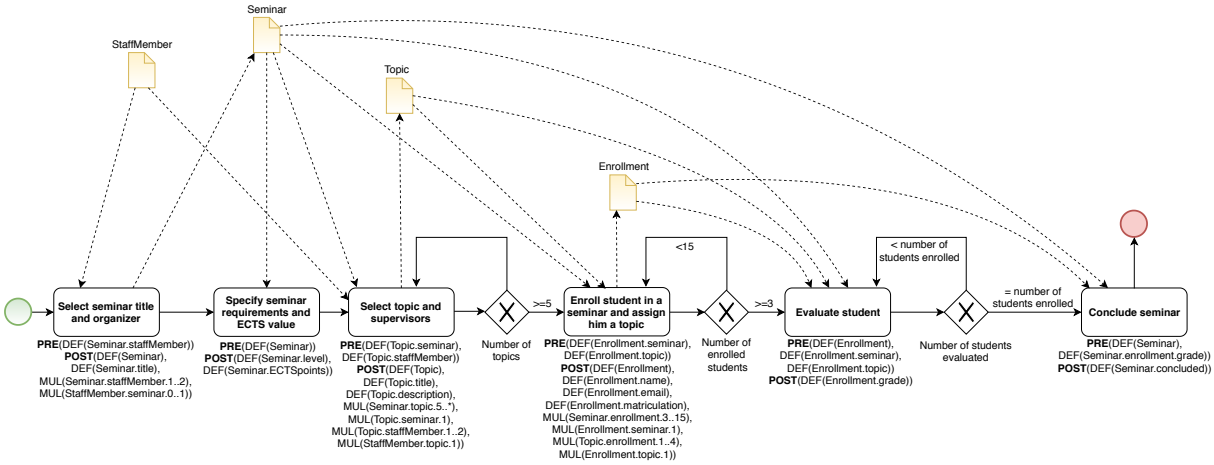


Figure 3.3: Activity model of the university seminar scenario.

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, hence `StaffMember` being used as the first activity's precondition. 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 [10]. 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.

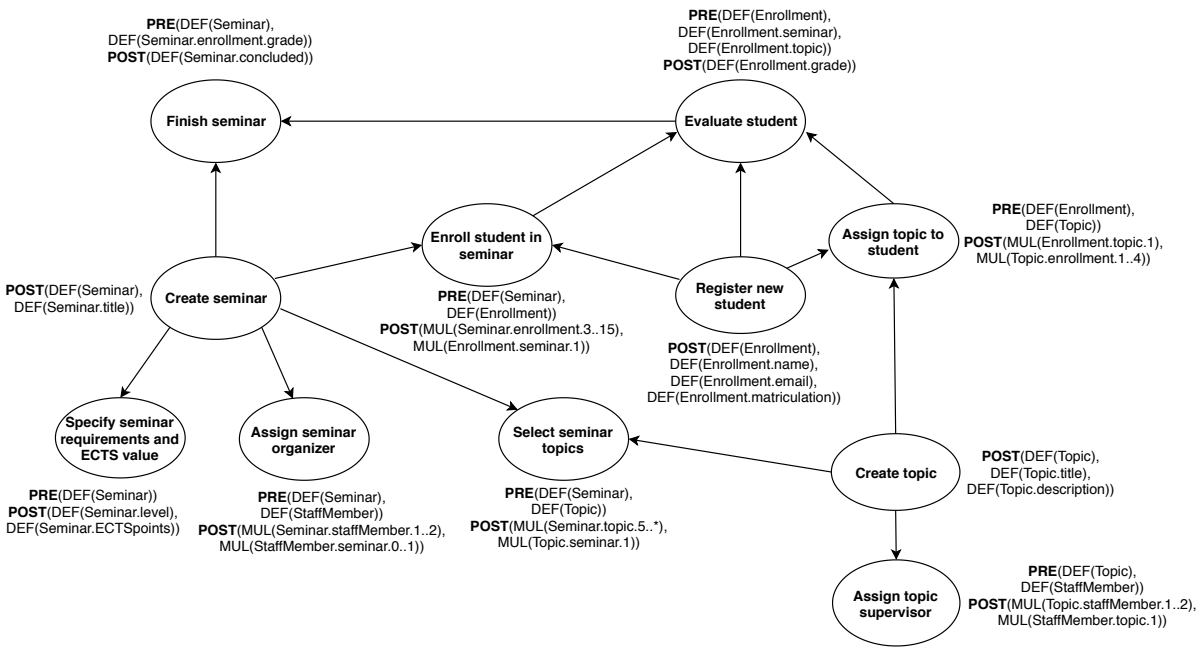


Figure 3.4: Goal model of the university seminar scenario.

3.4.2 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 3.4. 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.

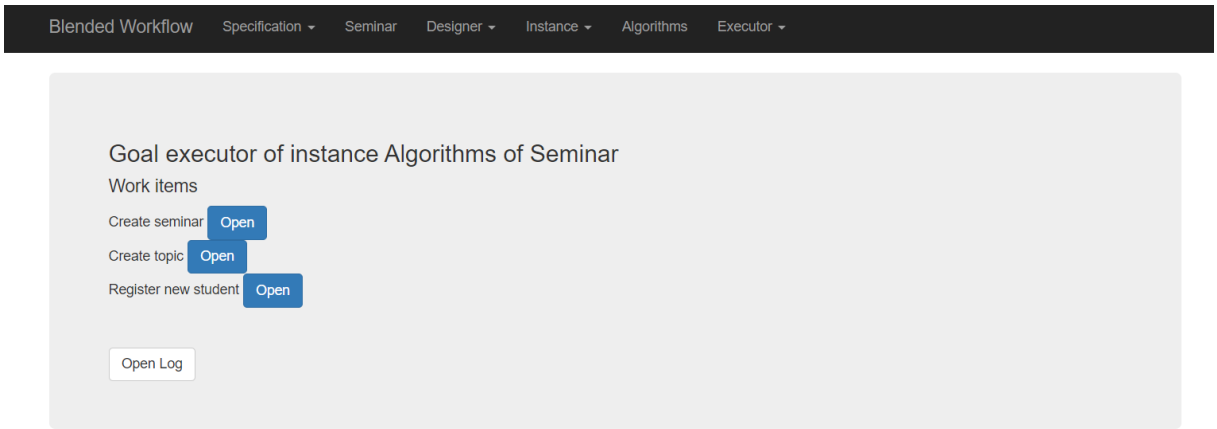


Figure 3.5: Blended Workflow's worklist of a university seminar process instance.

3.5 Execution Behavior

Blended Workflow follows a classical workflow execution procedure: when available, work tasks are placed in a worklist and workers can select a task from the worklist to execute. Performing a task changes the workflow state and conditions, thus enabling new tasks. In figure 3.5, we can see the goal model's initial worklist of a seminar process instance called "Algorithms". Notice that since it is the goal model view, there are three different work tasks available to start the process instance with.

What is not so orthodox in Blended Workflow is that it provides two different workflow approaches to perform the work tasks: the goal model and the activity model. In both models, the notion of sequence is confined to the work task's activation clause or precondition. If a given work task meets its activation clause, meaning that a set of requirements are in a valid state, then the work task is automatically activated for execution. If the work task has no activation clause (has no prerequisites), then it is allowed to execute from the beginning.

On top of the notion of activation clauses, the activity model's work tasks are also generated with an inherent order. That order reflects itself on the activation clause. If a particular work task `A` was generated to be performed before work task `B`, then the data elements defined in `A` will be present in the activation clause of `B`. This is why the activity model offers a smaller amount of execution paths than the goal model. In the goal model, the order in which the definitions are achieved is not so relevant, hence goals have fewer restrictions in terms of activation clauses. Because of this, usually a greater amount of goals can be chosen for execution by the users at any point in time. In simple terms, goals have fewer restrictions than activities; that is why the goal model is more flexible and choice-driven than the activity model.

Whether it is a goal or an activity, each work task is composed of preconditions (also previously mentioned as activation clauses) 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.

The work tasks that have valid preconditions and unfulfilled postconditions are displayed on the worklist, available for selection. Upon executing a work task, new data elements are instantiated and associations established. These new definitions will trigger the preconditions of the following work tasks, thus making them available for execution.

When an activity or goal executes, new data elements are generated according to the business process' data model. If a work task's postcondition is the definition of a new entity `E`, then, internally, a new object of entity `E` will be instantiated. If the work task just defines an attribute, it means that the entity the attribute belongs to is already defined. Therefore, within the work task, the user must select the "parent" entity instance from a list of candidates. Work tasks also define entity associations by selecting the entity instances that will be linked. The multiplicity of the association must be respected, meaning that if the association is one-to-one, only one entity instance of each entity is selected; if the association is one-to-many, one entity instance can be linked to many entity instances from the other entity, and so on.

As an example, take a look at the final activity of the university seminar scenario's activity model – "Conclude seminar". It has the following conditions:
precondition = {`Seminar, Seminar.enrollment.grade`},
postcondition = {`Seminar.concluded`},
which comprise 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`.

Internally, whenever a new operation is performed, Blended Workflow checks for process consistency by verifying if attributes are only defined for instances of their entity and have the correct data type, multiplicities are not exceeded, all associations are bidirectional, and dependencies are not violated. If any of these invariants is not respected, the operation is halted and an error message is displayed to the user, explaining what went wrong.

A given process instance concludes when its mandatory entity, as well as all its attributes, are well defined and the multiplicity (cardinality relation among data entities) and dependency conditions hold. Nevertheless, the goal model allows the execution to continue even after the definition of all mandatory elements. The work tasks that were not conducted during the execution can be performed later after the main goals of the workflow have been achieved. This is another reason why the goal model is so flexible. If desired, the user can go straight into the mandatory goals, and only then perform the optional ones.

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 [1, 6].

3.6 Execution Interchange Between Activity and Goal Model

Any process instance executing according to one of the workflow models can continue executing according to the other. 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 the goal model 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.

The execution order doesn't necessarily change when going from the goal model to the activity model, therefore dependency circularities are not introduced since it is ensured that the activity model doesn't contain any.

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.

3.7 Technical Details

Blended Workflow's implementation follows the Model-View-Controller (MVC) architecture pattern. The Model contemplates the domain objects, such as entities, attributes, and associations, among others. The View is the presentation layer where users interact with the system. Controllers are responsible for manipulating data, thus being the link between the Model and View.

The front-end runs on a web browser and acts as a client-side application that makes requests to the back-end and displays the responses. The front-end is developed using web technologies such as HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and Javascript. Currently, most of the front-end is implemented using the React library[2].

The back-end is responsible for managing internal data, which is stored in a database, and correctly replying to client requests. The back-end foundation is developed using the Spring framework[3] and the JAVA programming language. The communication between client and server is built on top of the Representational State Transfer (REST) Application Programming Interface (API) protocol, which uses basic Hypertext Transfer Protocol (HTTP) methods, such as POST and GET, to transmit messages and data. The communication is done through the use of Data Transfer Objects (DTO) in the JavaScript Object Notation (JSON) format.

[2] https://reactjs.org/
[3] https://spring.io/

4

Implementation of the Skip Operation

Contents

4.1 Introduction

The act of skipping something is characterized by passing to a forward point while omitting what was in between. Applying this concept to process management means that, for some reason, we want to have the ability to bypass a piece of work in order to immediately go to the next one. When a user decides to skip a work task, it is not considered standard behavior, hence being a mechanism designed to be used in special cases. Thanks to its loose nature and lack of flow rules, the goal model is the go-to approach when it comes to flexibility and dealing with unexpected situations. However, some users may be more accustomed to using a stricter approach like the activity model. For this reason, it would be beneficial to those users if they could have some kind of mechanism to handle exceptions. The ability to skip over activities helps to achieve that. The skip operation is a powerful mechanism for the activity model that allows skipping unwanted activities, as well as preparing the system for unexpected situations. These additions enhance the overall flexibility of the activity model. The skip mechanism fits the activity model philosophy since it maintains the notion of sequential progression. Even when activities are skipped, the flow of the process keeps moving forward. By its own nature, the goal model allows the user to choose what to do next, therefore the ability to skip goals is not as significant as in the activity model since the user can decide if a goal needs to be executed or not. Besides, introducing the skip operation in both models could mean that some definitions would never be achieved regardless of the model (for instance, both the activity and goal of an entity definition could be permanently skipped), which could compromise process completion since. For these reasons, we decided to apply the skip operation just in the activity model.

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 [11]. 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.

Skipping activities allows users to save time and effort for actions that are not needed in a particular case, but it comes at the cost of raising data-related issues. In this chapter, 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.

4.2 Internal Management of the Skip Operation

When it comes to understanding our Blended Workflow skip implementation, there is a property that is important to highlight: activities are atomic. This means that 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. Partial execution of activities is not permitted.

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 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.

4.3 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 [11, 12] 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.

Consequently, during Blended Workflow's execution time, every entity instance and attribute instance that has been instantiated is in a valid state. There are three possible states: *Undefined*, *Defined*, and *Skipped*. The possible state transitions are demonstrated in figure 4.1. 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.

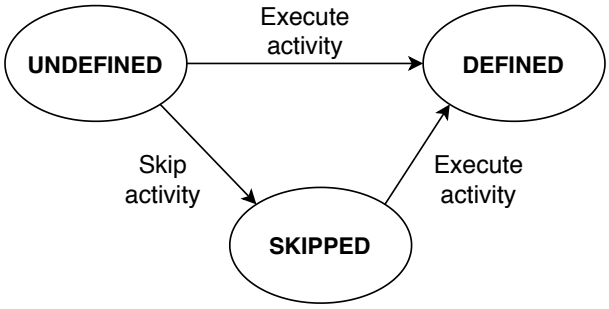


Figure 4.1: State transitions of data elements.

4.4 Behavior of the Skip Operation

Earlier we discussed the data model of a Blended Workflow business process and distinguished between mandatory data and non-mandatory data. Each data model has one mandatory entity and all its attributes are also mandatory. 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 4.3. Its

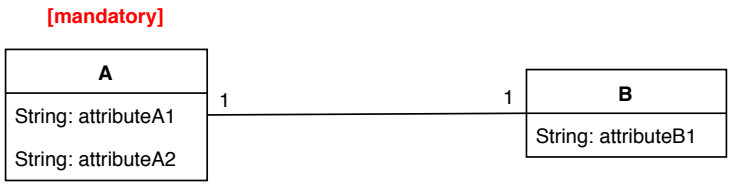


Figure 4.2: Data model with two entities.

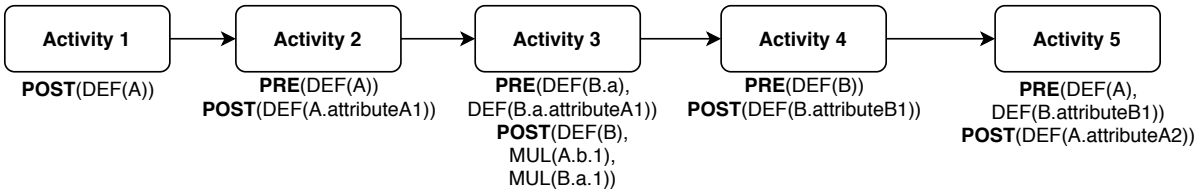


Figure 4.3: Activity model with two entities.

data model, represented in figure 4.2, 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. This will be the entity instance the attribute belongs to. 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.

The skip functionality is also applicable in activities with multiple attribute definitions, as long as none of these attributes is mandatory. These are activities that define various attribute instances in a single work task. The attributes may belong to the same entity, or they can be part of different parent entities. If this kind of activity is skipped, then all the attributes it holds are set to the *Skipped* state. These attributes may or may not have dependencies with other attributes.

4.5 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. It does not make sense to finish the seminar if some students have not been evaluated yet. In the data model of the business process, these dependencies must be explicitly modeled by the designer to enforce the workflow behaves as expected.

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.

4.5.1 Dependency Tree

We developed a mechanism called dependency tree whose function is to assist the management of attribute dependencies. The dependency tree has two main goals. One of them is to support the awareness and understanding of the dependency chain of a given attribute instance. This is done by providing a visual tool (developed using the *react-archer* library[1]) that is particularly helpful when there are several attribute instances involved in the dependency chain. As we can see by figure 4.6, dependency chains often tend to shape as a tree-like graph, hence the name "dependency tree". 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 – an informed 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, 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

[1] https://www.npmjs.com/package/react-archer

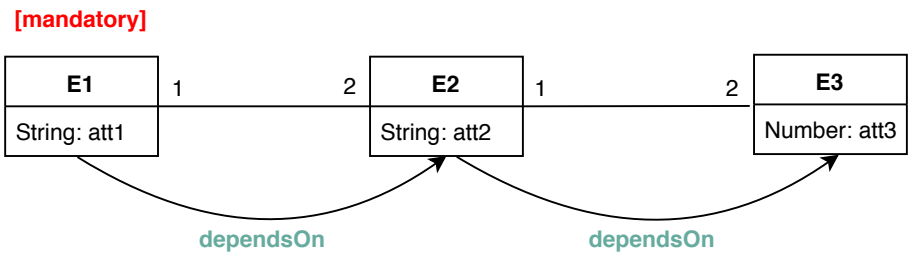


Figure 4.4: Data model with three entities.

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. The parent entities of these attributes are not relevant. They may all belong to the same entity, or to completely different entities. Dependency trees correctly display the parent entity of each attribute instance.

Dependency trees are only displayed when the selected attribute instance has dependencies with other attribute instances. Since one of its purposes is to aid the dependency visualization, dependency trees can be presented whether the selected attribute instance has all dependent attribute instances in the *Defined* state, or some in the *Skipped* state.

Consider the data model pictured in figure 4.4. It has three entities: `E1`, the mandatory entity, `E2`, and `E3`. Each entity contains one attribute. Attribute `att1` belongs to entity `E1`, attribute `att2` belongs to entity `E2`, and attribute `att3` belongs to entity `E3`. This data model has two explicit dependencies: `E1.att1 dependsOn E1.e2.att2`, and `E2.att2 dependsOn E2.e3.att3`. This is a dependency chain which declares that `att1` is dependent on the value of `att2`, which in turn depends on the value of `att3`. On top of that, the association multiplicities will enforce that to fill in `att1`, the values of two instances of `att2` (the associated ones) are provided. Similarly, each instance of `att2` requires the values of two associated instances of `att3` as well.

The activity model of this data model is presented in figure 4.5. It is composed by three activities: Activity 1 defines non-mandatory entity `E3` and its attribute `att3`; Activity 2 defines non-mandatory entity `E2`, its attribute `att2`, and the association with `E3`; Activity 3 defines the mandatory entity `E1` and its attribute `att1`, as well as the association between `E1` and `E2`.

Let's suppose the user executing the workflow decides to skip Activity 1 and Activity 2. Instances of `E3`, `E3.att3`, `E2`, and `E2.att2` go to the *Skipped* state, and the attributes `E3.att3` and `E2.att2` have no

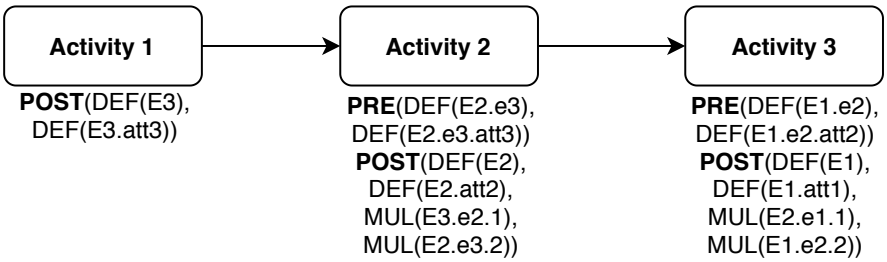


Figure 4.5: Activity model with three activities.

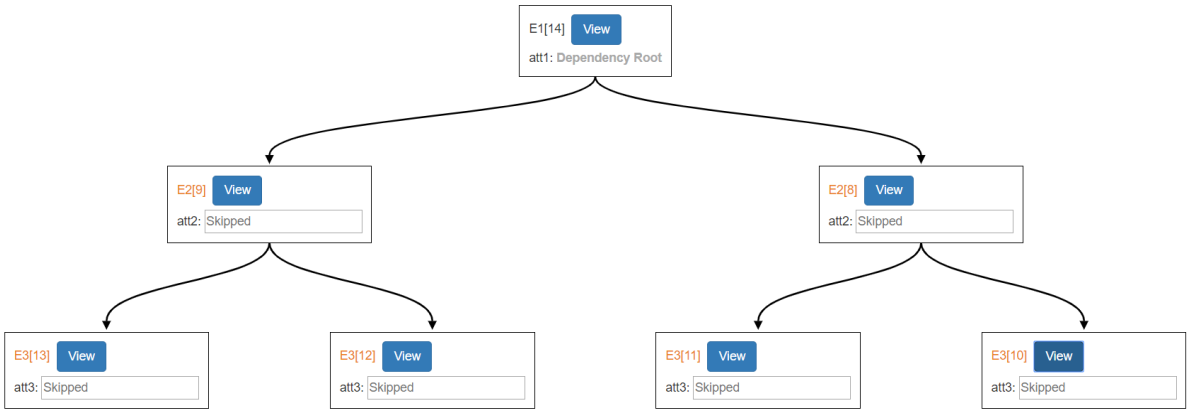
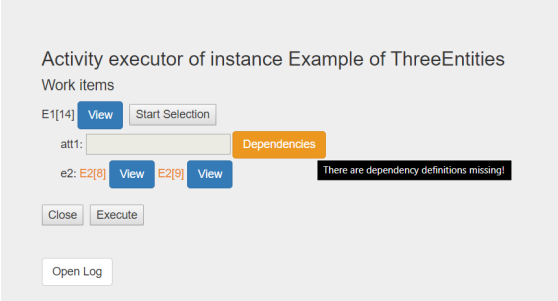


Figure 4.6: Dependency tree with missing values.

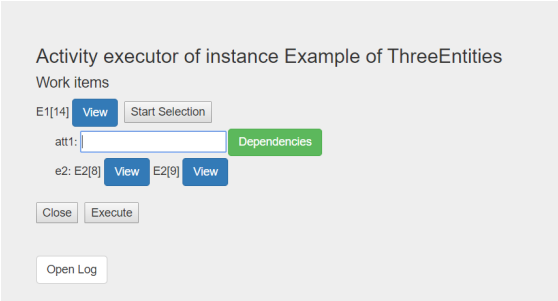
values. Activity 3 must be executed as it holds mandatory data. However, since the definition of `E1.att1` requires that `E2.att2` is defined and with a specific value, the user has to utilize the dependency tree to assign a specific value to `E2.att2`. And since the value of `E2.att2` depends on the values of the instances `E3.att3`, those instances will have to be defined in the dependency tree as well. Figure 4.6 shows the dependency tree of the attribute `E1.att1`. Note that, at this point, the attribute instances do not have values yet; those values should be provided here.

By providing those values, the entity instances `E2` and `E3`, and the attribute instances `E2.att2` and `E3.att3` go to the *Defined* state. Since the dependent attributes now have specific values, the attribute `E1.att1` can be defined, and the activity's postcondition achieved. Figure 4.7 displays Activity 3's user interface before and after performing the skipped attribute instances definitions in the dependency tree. In figure 4.7(a), the skipped attribute instances in the dependency chain have not been defined yet, therefore the user is not able to introduce a value for attribute `E1.att1`. But after using the dependency tree to perform such definitions, all the dependency instances become defined and `att1`'s field is then activated for user input, as we can see in figure 4.7(b).

Notice that by defining previously skipped data instances, we are performing more actions than what is suggested by Activity 3's postcondition. We are not only achieving the postcondition of Activity 3 but also redefining some extra data elements from the first two activities' postconditions. This always hap-



(a) Before dependency tree definitions.



(b) After dependency tree definitions.

Figure 4.7: Activity 3's user interface before and after the dependency tree attributes definitions.

pens 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.

If within an activity, previously skipped attribute instances are redefined using the dependency tree and after doing so, the user decides to skip the activity anyway, then those attribute instances definitions will not be saved internally. We enforce this behavior because the information provided in the dependency tree would only be useful for the data within the activity (the dependent data). If the activity is skipped, it means the information in the dependency tree did not contribute to a decision, thus it should be discarded.

Any dependency tree can adequately scale horizontally and vertically. This means that whether there are one or several attribute instances within a horizontal row of the dependency tree, the user will be able to visualize all of them. Additionally, if the dependency chain consists of one dependency or many, the dependency tree window can be scrolled down to check all of them in terms of depth.

Upon providing the values for skipped attribute instances in the dependency tree, if any of those values do not match the correct data type for that particular attribute, an error message is displayed to the user, explaining the error and which attribute instance is at fault. For instance, if the user inserts a String for the attribute "Student age", the value inserted will not be accepted, as a Number data type is expected. A similar error message is presented if no value is provided for an unfulfilled attribute instance.

The goal model also incorporates the dependency tree mechanism. When a goal defines an attribute that has dependencies, its dependency tree can be visualized. The dependency tree is not stranded in the activity model because it has the general purpose of aiding the users' visualization and comprehension of attribute dependencies, as well as redefining the skipped attribute instances in the tree. Therefore, the dependency tree is helpful, regardless of the model. In the next section, we will see how the goal model offers another option to redefine such instances.

4.6 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 example also shows that skipping an activity doesn't necessarily mean we are ignoring it. We may skip a work task because, at that moment in time, we might not have the necessary information to execute it, but we already have enough information to perform subsequent work tasks. We may also skip an activity because we might want to delay a decision or result to the future. 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 model and the goal model, consider the activity model depicted in figure 4.5 again. Remember that entity `E1` is mandatory. Let's assume the user decides to skip Activity 1 and Activity 2 because they only have non-mandatory data elements. In the activity model, Activity 1 and Activity 2 will not be displayed again because they were skipped, but the goal model will maintain the goals that achieve Activity 1 and Activity 2'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 instances `E2` and `E3`, and the attribute instances `E2.att2` and `E3.att3` which were instantiated as *Skipped* in Activity 1 and Activity 2. By doing so, when Activity 3 comes, there is no need to define anything using `E1.att1`'s dependency tree, since all previously skipped instances were already redefined using goals.

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.

4.7 Execution Log Integration

Blended Workflow comes equipped with an execution log that allows users to audit the current process instance. Checking a process instance's history can be a valuable mechanism that provides a better understanding of the process execution steps. The execution log offers information regarding the order of task execution, task nature, task information, and the tangible results of executing a task. It can also display information regarding the identity of the user who performed the task, though this functionality was disabled for our work.

An example of a process execution log can be seen in figure 4.8. We can observe that the first task, called "Enroll student", created an entity instance of entity `Enrollment` with the internal id 16. The second task was an activity named "Insert student email" that used the entity instance from the first task as precondition, and produced the attribute `email`. The value of this attribute is set to "SKIPPED", which means the activity was skipped. The following task (task number 3), was a goal that produced a defined value for the previously skipped attribute. This reflects the behavior discussed previously, where a goal redefines a skipped attribute instance. By looking at the execution log, it is possible to understand the flow of the process execution.

4.8 Impact of the Skip Operation on Final Goal Achievement

In this chapter, we have seen the overall behavior of the new skip operation. We saw how skipping activities can affect future activities, and how the issue of missing data is handled. But this new functionality begs the question: what is the impact of skipping all this information input, and is it always possible to achieve the workflow's final goal? The answer is: yes, even with the ability to skip most information input, the final workflow goal is always achievable. Since Blended Workflow differentiates between mandatory and non-mandatory data, and we impose that the mandatory data cannot be skipped, then as long as the designer identified the mandatory data accurately, the workflow should correctly support

Activity executor of instance Algorithms of Seminar

Work items

Enroll student Open

Evaluate student Open

Insert new topic Open

Insert student name Open

Insert topic description Open

Close Log

Order	Operation Name	Operation Type	Preconditions	Products	Execution User
3	Insert student email	Goal	Enrollment[16]	Enrollment[16,email[andrepr@email.com]]	
2	Insert student email	Activity	Enrollment[16]	Enrollment[16,email[SKIPPED]]	
1	Enroll student	Activity		Enrollment[16]	

Figure 4.8: A process instance's execution log.

the behavior of the process. Mandatory data is considered the most important data; the crucial data that characterizes the business process. If this kind of data was allowed to be skipped, then the process would miss its essence and purpose. Therefore, by enforcing that this data should always be provided in every process instance, we guarantee that the workflow objective will be always attainable. Even in situations where mandatory data depends on other missing data (previously skipped data), the process flow is only allowed to continue once the missing data is fulfilled, using the goal model or the dependency tree, thus enabling the definition of the mandatory data.

The act of skipping allows a process instance to become minimal in terms of the information it holds, but for the sake of process consistency, some information is always required to be fulfilled, regardless of any other condition or exception situation.

4.9 Seminar Organization Scenario – Skip Operation

In this section, we will consolidate the behavior of the skip operation by looking back to the university seminar organization scenario and see how the skip action is applicable. In the activity model of the university seminar scenario pictured in figure 3.3, the first two activities – "Select seminar title and organizer" and "Specify seminar requirements and ECTS value", as well as the last one – "Conclude seminar", are the only unskippable activities. These activities define the mandatory entity `Seminar` and its attributes, thus containing crucial information for the business process that cannot be skipped. All other activities in between only define non-mandatory data elements in terms of entities and attributes, hence being skippable.

Even though these activities can be skipped, the associations they define should be set. By skipping those activities, the non-mandatory entities and attributes go to the *Skipped* state, but the association between entities, even if in the *Skipped* state, must be defined.

Skipping the activity "Evaluate student" leads to having an undefined or skipped value for the attribute `grade` of entity `Enrollment`, and as we have seen previously, in order to set the attribute `concluded` from entity `Seminar`, the `Enrollment.grade` attribute needs to be defined first because there is an explicit dependency between them. Therefore, if the "Evaluate student" activity is skipped, the "Conclude seminar" activity cannot be performed. To enable the execution of the last activity, we need to define a specific value for the attribute `Enrollment.grade`. There are two ways to achieve that.

The first one is by using just the activity model. Upon selecting the "Conclude seminar" activity, the user will see that there is a dependency definition missing. Using the dependency tree, the user can reset a value for the `Enrollment.grade` attribute that was previously skipped, thus enabling the definition of the final attribute `Seminar.concluded`.

The other option requires switching the execution to the goal model, and performing the goal that defines the `Enrollment.grade` attribute. Since the goal model allows us to define data elements that were skipped in the activity model, the user will be able to redefine a value for `Enrollment.grade`, and this will enable the execution of the final activity.

Both ways achieve the same end of making the final activity enabled for execution, but since Blended Workflow provides two workflow models, the user can choose two different options to achieve that objective.

5

Comparative Analysis with Other Approaches

Contents

5.1 Introduction

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.

5.2 Case Handling

One of the most popular methods to deal with change and volatile environments is the case handling approach [2, 12]. 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.

5.3 Product-Based Workflow Design

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 (PBWD) approach [13], 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 PBWD 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 PBWD, 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 PBWD, 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 PBWD'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, PBWD'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.

5.4 Other Notorious Approaches

The work of [3] 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.

The case management approach [10] introduces the concept of fragments, which are structured pieces of work, and uses them in the workflow's modeling phase, resulting in the possibility of having multiple valid execution paths. In case management, activities can only be skipped if, upon an XOR gateway split, the alternative path is chosen for execution. In this situation, the activities on the path that wasn't chosen may be skipped. Though this is not the kind of explicit skipping behavior we were looking for, it is an insightful one to take into account.

In the approach suggested in [11], 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.

Other object-aware approaches like PHILharmonicFlows [14] do not have an explicit mechanism that allows skipping activities. Instead, the user can implicitly skip optional tasks by not executing them. This is also how it is done in CMMN [15, 16], which is the standard to model and represent declarative business process workflow implementations. Blended Workflow's goal model shares this behavior.

In [17], skipping is seen as an internal fault handling mechanism. This means that whenever a fault

occurs and consequently blocks an activity, some workflow implementations might handle those exceptions by skipping the current activity, going straight to the next one so that the process execution continues. In Blended Workflow, activities can be skipped in exception situations which does not necessarily mean there was an error. The process execution is never permanently blocked in Blended Workflow. In rare cases where the execution blocks, the user can switch to the other model and is assured to be able to continue using that model.

5.5 Value Proposition of Our Approach

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.

6

Conclusion

Contents

6.1 Conclusions

The application of an efficient workflow is fundamental to have a thriving business process management foundation in today's working landscape. There is no doubt that an effective workflow is at the heart of productivity in any workplace. Companies, big and small, use different types of workflow implementations to control resources, manage personnel, and accomplish their goals. Flexibility is an important workflow characteristic in knowledge-intensive environments, where exceptions and unexpected situations may occur frequently, and workers are more predisposed to guide the process based on their experience. Therefore, a workflow capable of adapting to the situation in hand is pivotal to success.

In this thesis, we dive into the notion of flexible workflows and apply a flexible mechanism – the skip operation – to a naturally rigid imperative workflow approach. Skipping data input may raise some issues of missing information, such as process completion and data dependencies, but by taking advantage of the dependency tree, and Blended Workflow's declarative model, we can mitigate those issues. A part of our skip implementation solution is specific to Blended Workflow, in the sense that we take advantage of its duality in terms of execution environments, by using the declarative goal model to compensate the skipped information in the imperative declarative model. However, the key to our skip implementation is the notion of data elements state, and this may be generalized to most imperative workflow approaches. In the process of designing our solution, we came up with the dependency tree mechanism, which ended up revamping the way users can visualize and understand attribute dependencies in Blended Workflow.

Lastly, in chapter 5, we looked into other credible approaches, explained how they achieve the skip operation, and compared their implementations to our own.

The code of Blended Workflow, including the skip operation, is publicly available in a GitHub repository[1].

6.2 Main Contributions

Even though the benefits of having a flexible workflow environment have been widely known for a while, the implementations of such flexible approaches tend to come almost exclusively in declarative workflow methodologies, as discussed in chapter 5. With our work, we propose and implement a way to apply and achieve flexibility in an imperative workflow approach, which are notorious for their natural rigidity and lack of flexibility. In doing so, we allow workers that are accustomed to imperative workflow methodologies to skip over undesired activities, thus deviating the workflow execution from the predetermined specification. 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, our skip operation makes a stricter imperative workflow specification more flexible.

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

6.3 Future Work

There are a few features that could be improved upon in the future. Throughout this work, we tried to make every component of Blended Workflow compatible with the new skip operation. We managed to achieve that in almost every aspect, except in the execution log. Though the execution log supports the skip operation, there is a specific case where it does not reflect the correct happening of events. When an attribute instance is skipped and later it is redefined using a dependency tree in the activity model, the initial skipped log entry is updated to reflect the new value. This is not the desired behavior, because the log is not being coherent with what actually happened in reality. The skipped log entry should remain, and a new entry should be added with the redefined attribute instance value. This detail was not implemented because to do it, the entire log implementation would have to be restructured from scratch, and it would have taken a lot of effort.

In terms of new features, the concept of user roles could be added in the future. As discussed in chapter 5, user roles are useful in this kind of environment as they allow authorization management. At the moment, every user is capable of skipping activities. If user roles were applied, only users with proper authorization would be able to perform this kind of action.

When it comes to enhancing Blended Workflow's flexibility capabilities, an obvious candidate is adding the redo operation. With our work, we allow users to skip some undesired tasks. With the ability to redo tasks, users could go back and perform a previously executed task. This type of mechanism is helpful in scenarios where we want to revert mistakes or decisions. It would be another flexible operation that would make Blended Workflow better prepared to deal with exceptions.

Bibliography

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

[2] 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.

[3] 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.

[4] N. Mulyar, M. Pesic, and M. Peleg, “M.: Towards the Flexibility in Clinical Guideline Modelling Languages,” *Information Systems Journal*, pp. 1–18, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.817

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

[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. Reichert, “Enabling Flexible and Robust Business Process Automation for the Agile Enterprise,” pp. 203–220.

[8] G. Regev, P. Soffer, and R. Schmidt, “Taxonomy of Flexibility in Business Processes,” pp. 90–93.

[9] I. Beerepoot, I. V. D. Weerd, and H. Reijers, “Business Process Improvement Activities : Differences in Organizational Size , Culture , and Resources.”

[10] 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.

[11] 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.

[12] 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.

[13] 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. [Online]. Available: https://www.jstor.org/stable/40398622

[14] C. M. Chiao, V. Künzle, and M. Reichert, “Object-aware process support in healthcare information systems: Requirements, conceptual framework and examples,” *International Journal on Advances in Life Sciences*, vol. 5, no. 1-2, pp. 11–26, 2013. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84880688910&partnerID=40&md5=e9a02835179331a326baada3eca92852

[15] M. Marin, R. Hull, and R. Vaculin, “Data Centric BPM and the Emerging Case Management Standard : A Short Survey Case Management Modeling and Notation (CMMN): An emerging OMG standard,” vol. 257593, no. September, pp. 24–25, 2012.

[16] M. A. Marin, “Introduction to the Case Management Model and Notation (CMMN),” pp. 1–23, 2016. [Online]. Available: http://arxiv.org/abs/1608.05011

[17] S. Wu, “A New Method of Exception Handling in Workflow,” *2009 International Symposium on Intelligent Ubiquitous Computing and Education*, pp. 420–422, 2009.