# MIS Module
# Workflow Management
# V2.0

Requirements Specification

| | |
|---|---|
| **Project:** P-RC-01-03 Workflow Management V2.0 | **Reviewed by:** Dragan Milicev |
| **Document type:** Requirements Specification | **Review date:** July 31, 2001 |
| **Document version:** 1.0 | **Comments:** |
| **Author:** Marko Zaric | |
| **Date:** July 2001 | |

# Introduction

This document contains general requirements specifications for the MIS Module *Workflow Management*, Version 2.0. It describes the general concepts that lie beneath a common workflow management strategy, and offers a solution for the implementation of these concepts in MIS.

Workflow management is concerned with the automation of work processes in which information, documents, or any other objects are passed between participants according to a defined set of rules to achieve a predefined goal. A workflow is usually defined for the needs of a certain organisation whose work or business processes have reached a certain complexity, so that they can no longer be manually controlled. Participants in a workflow are usually the company's employees, although, in some cases, work can be performed by computer applications too.

A well-defined workflow has the structure of a graph, where the nodes of the graph represent amounts of work to be performed by workflow participants, and the edges specify the flow of work from one participant to another. A work process, therefore, consists of a network of activities that are performed in some order, depending on conditions and events that occur during the execution of the process. Various resources may be produced, transformed or used during the execution of an activity, and the performer of the activity has the control over the usage of those resources. Activities may also contain criteria that indicate the start and termination of the process, or some other information, such as the duration of an activity, etc.

A distinction should be made between a workflow definition and its implementation. The workflow definition acts as a template for the creation and control of instances of that workflow, during workflow enactment. Therefore, the abstractions, described in this document, that refer to the workflow definition fall into the "Type" side of this type-instance dichotomy, and the abstractions that refer to the implementation fall into the "Instance" side.

This document is organized as follows. The first chapter describes the structure of the MIS module Workflow Management, and gives the detailed explanation of its main concepts. The chapter is dived into two sections. The first section deals with the definition (type) side, and the second with the implementation (instance) side. The second chapter explains the behavioral concepts of the MIS module Workflow Management by presenting the possible use cases that could be found in practice. It also discusses the implementation features of the presented use cases. The addendum gives a complete overview of the functionality provided by the module Workflow Management V2.0, as well as of the enhancements that the module introduces in regards to the Workflow Management V1.0.

Beside the same usual conventions, used in other documents with requirements specifications, this document will also use the following conventions. Namely, there will be a slight inconsistency between the names of abstractions described in the document and the names of the same abstractions defined in the model. To accentuate the fact that all the abstractions belong to the Workflow Management module, their names in the model will start with the prefix *WF*. On the other hand, for the reasons of clarity, these abstractions will be referred to in this document without the prefix. Also, the names of abstractions defined on the "Instance" side in the model will end with the postfix *Ins*, but they will be referred to in the document with the full word *Instance*.

The actions, such as creation of instances and links, of classes and associations specified at the definition side, are said to be performed at “design-time,” and the actions on instances and links at the implementation side will be performed at “run-time.”

# Structure

## Workflow Definition

*General Concepts*

The main concept in workflow management is the concept of an *Activity*. An Activity represents work that will be processed by a workflow participant or a computer application. As it was previously mentioned, a workflow consists of a network of Activities that are related to one another via transition information. The relationship between Activities, showing which Activity will follow another, and therefore specifying the order of Activities in a workflow, is captured in the concept of a *Transition*. Each Transition starts from, and ends at exactly one Activity. In a general case, an arbitrary number of Transitions may start from an Activity, and an arbitrary number of Transitions may end at an Activity.

An Activity can either be an atomic Activity (*Action*), in which case it represents the smallest amount of work, or a composite Activity (*Process)*, meaning that it could be further decomposed into smaller Activities. An Action represents the smallest unit of self-contained work. Other optional information may be contained within the Action, such as information on whether it is to be a starting/finishing Action within its parent Process, the minimal, maximal and expected duration of the Action, etc.

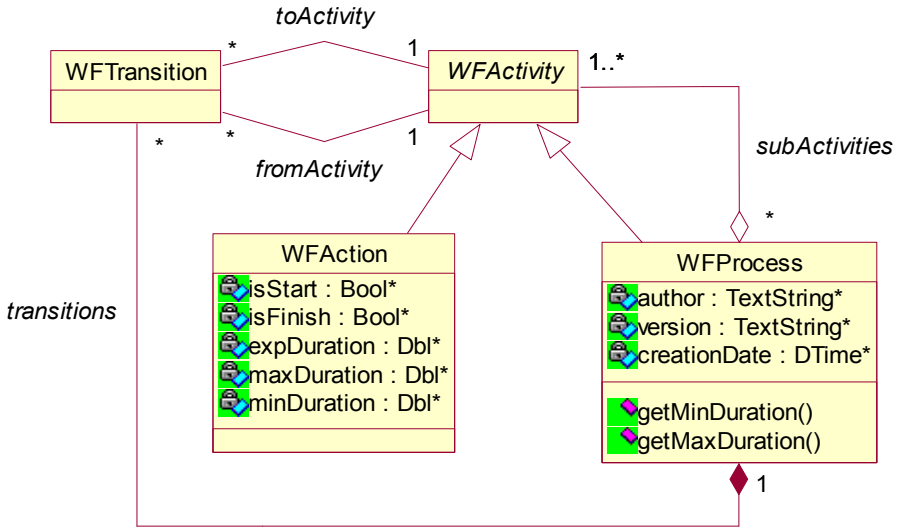


**Figure 1:** Definition side - General concepts

A Process consists of one or more Activities. A specific Activity may be a part of several Processes, i.e. the work represented by the Activity can be performed in several different Processes. Transitions between Activities in a Process, however, are part of that Process only, i.e. their scope is local to the specific Process definition. A Process provides information

associated with the administration of a process definition (author, version, date of creation, etc.). Process also provides the functionality to calculate the shortest/longest route of all the Activity routes that can be traversed during Process execution.

In some special cases, Activities perform no work processing, but simply support routing decisions within the incoming transitions and/or within the outgoing transitions. These special cases include Fork, Join, and Branch Activities. *Fork* and *Branch* Activities must have exactly one incoming Transition and more than one outgoing Transitions. When a Fork Activity is initiated, the workflow is split, so that the work may be processed in several Activities concurrently. In case of a Branch Activity, only one outgoing Transition can be fired. The decision about the Transition to be fired is made by evaluating a specific condition. The definition of a Brach Activity's condition may be given in form of a Script, represented by the abstraction MSScript. Conditions and Scripts will be explained with more details in the next section. As for the *Join* Activity, it must have exactly one outgoing Transition and more than one incoming Transitions. In case a Join Activity is initiated, all the Activities that precede the Join Activity must be completed, so that the outgoing Transition can be fired.

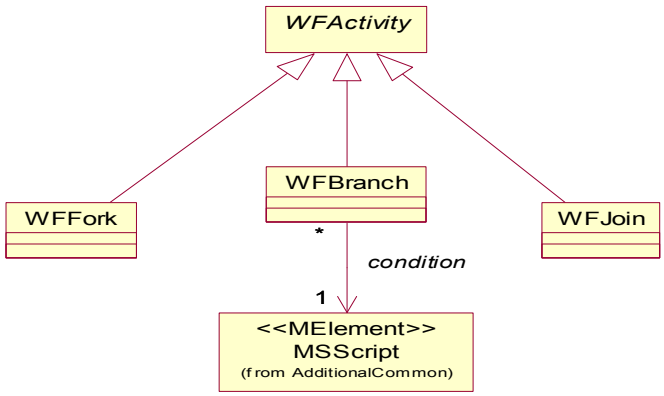


**Figure 2:** Fork, Join and Branch Activities

*Conditions*

Transition from one Activity to another may be conditional (involving expressions which are evaluated to permit or inhibit the Transition) or unconditional. If a Transition is conditional, the expressions for the evaluation of the condition may be given in form of a Script, represented by the abstraction MSScript. The textual value of the attribute *sourceCode* may actually be an operation specified in a scripting language, such as VBScript or JScript. The operation is called each time a certain condition is evaluated.

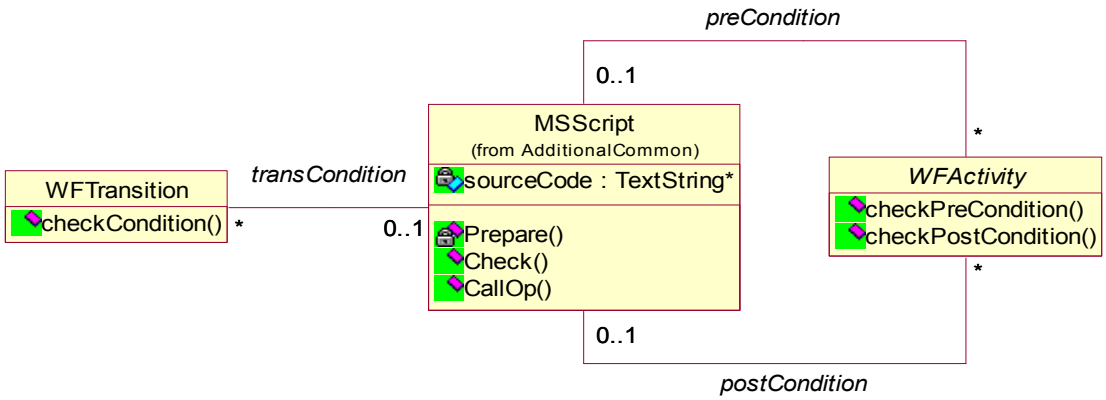


**Figure 3:** Activity and Transition Conditions

An Activity may have pre/post conditions. If a pre condition is specified for an Activity, then the Activity cannot be started unless the condition is satisfied. If a post condition is specified for an Activity, then its outgoing Transitions cannot be fired unless the condition is satisfied. Pre or post conditions for an Activity may also be defined in form of a Script.

*Roles*

A workflow participant performs partially or fully the work represented by a workflow Activity. A workflow participant is usually a person employed in an organisation, and, therefore, has a certain role in it. This relationship is represented in the module *Organisation Structure*, by the association between the classes *OrgUnit* and *RoleInOrg*.

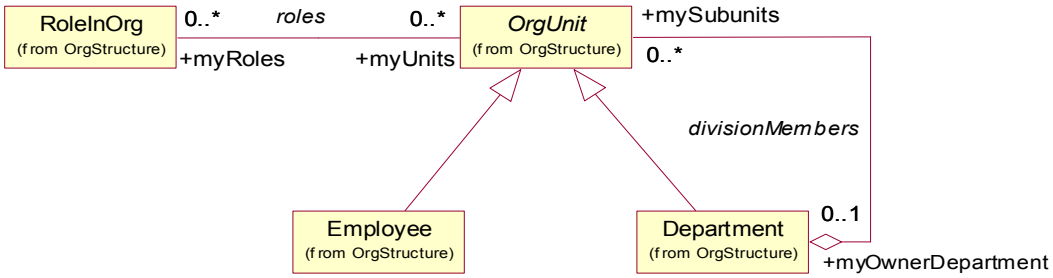


**Figure 4:** Roles in Organisation Structure

The Role in Organisation may specify a set of attributes, qualifications, and/or skills of a workflow participant. The Role, therefore, provides description of the participant that can act as the performer of an Activity. Associating a Role with an Activity (at design-time) would mean that only the participants with the specified Role might perform that Activity's work at run-time. More than one role may be associated with an Activity.

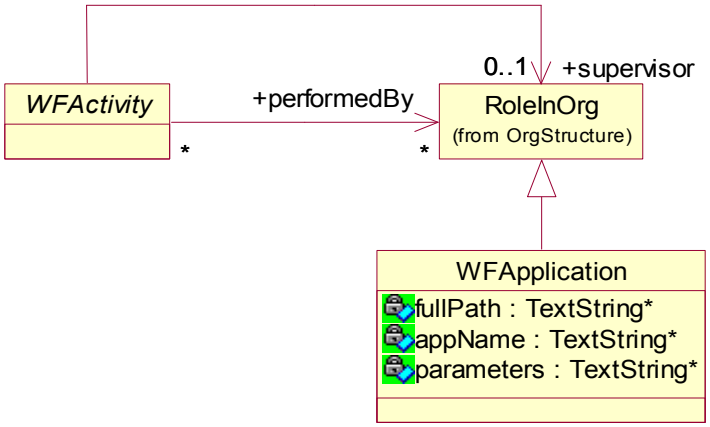


**Figure 5:** Activity Roles

The work represented by an Activity is always performed at run-time under the supervision of another Role, i.e. each workflow participant has a supervisor. That would require that a Role for the supervisor might also be specified at design-time. This relationship is represented by another association between the Activity and the RoleInOrg abstractions.

An Activity may be a manual Activity or an automated Activity. That means that an Activity can either be manually performed (by a participant), or an application can support the processing of data in order to accomplish the objective of the Activity. The application may be invoked by the workflow service to support, or wholly automate, the processing associated with the Activity. The application is represented by the abstraction *Application*, derived from RoleInOrg. The Application abstraction specifies the name and the full path of the application to perform the Activity, as well as any parameters to be passed to the application.

*Resources*

During the execution of an Activity, a workflow participant creates, uses, or changes different resources. The resource data is contained within MObjects that are either created or modified at run-time. The specification of the types of MObjects that can be used during the execution of an Activity, however, is given at design-time. The abstraction *Resource* is responsible for this specification. The attribute *resourceData* holds the information about the type of MObject that can be created/modified. It may also hold additional information about the specific actions that can be performed on the MObject. For example, it may specify which properties may be modified, and which properties may only be viewed. This information is presented in form of a specially formatted string that can be read at run-time. A Resource may also represent a concrete MObject, meaning that, at run-time, the same MObject would have to be used/modified.

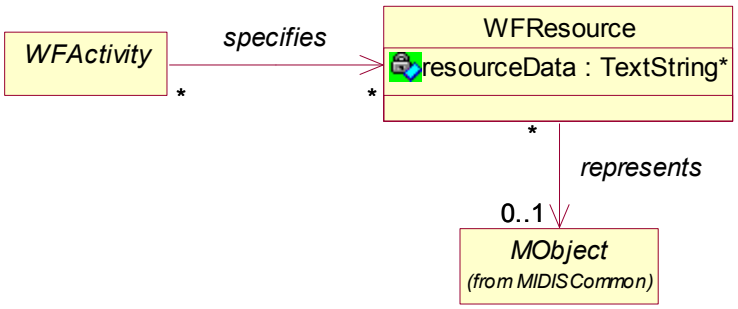


**Figure 6:** Resources

# Workflow Implementation

All of the concepts at the definition (type) side, as well as the relations between them, have their counterparts at the implementation (instance) side. Thus, the structure of this chapter will follow the structure of the previous, and the corresponding concepts will be presented in the same order.

*General Concepts*

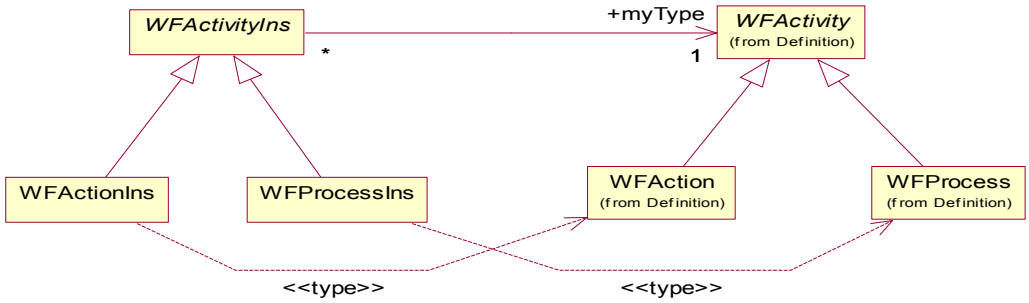


**Figure 7:** Type-Instance dichotomy

The abstraction *Activity Instance* is the instance-side equivalent of the Activity abstraction. It represents the concrete run-time incarnation of the Activity defined at design-time. The classes derived from Activity also have instance-side incarnations in the classes *Action Instance* and *Process Instance.*

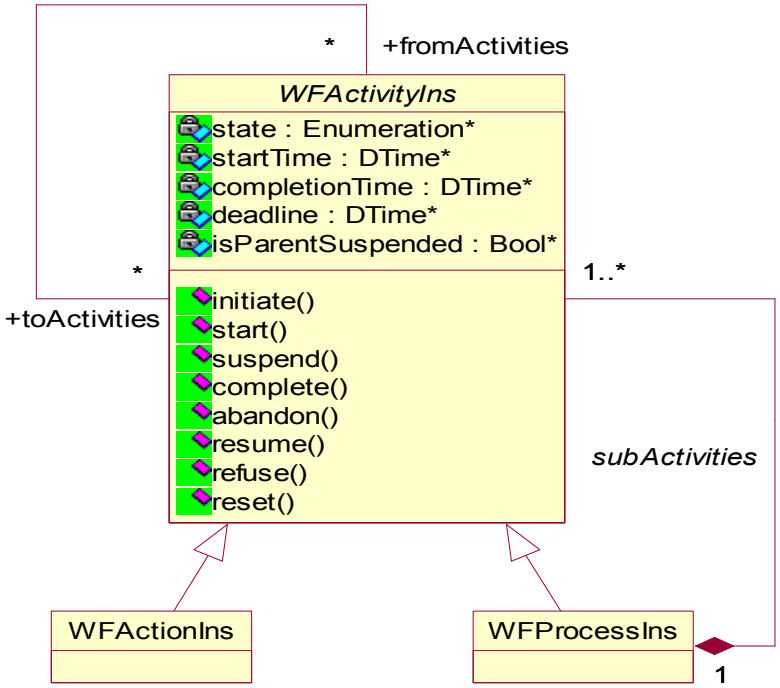


**Figure 8:** Implementation side – General Concepts

The network of Activity Instances belonging to a Process Instance is built upon the internal structure of the Process Instance's Process definition, i.e. upon the structure of the Process' sub Activities and Transitions between them. When instantiating a Process, the internal structure of the Process' Activities and Transitions is traversed, and the corresponding Activity Instances are created and linked accordingly (Figure 9). Two Activity Instances will be linked if there is a Transition, on the definition side, that connects their Activity types. A Process Instance will have as many Activity Instances as its Process definition has Activities. Contrary to the definition side, where an Activity may belong to several Processes, an Activity Instance must belong to exactly one Process Instance. It is created when the Process Instance is created and destroyed when the Process Instance is destroyed.

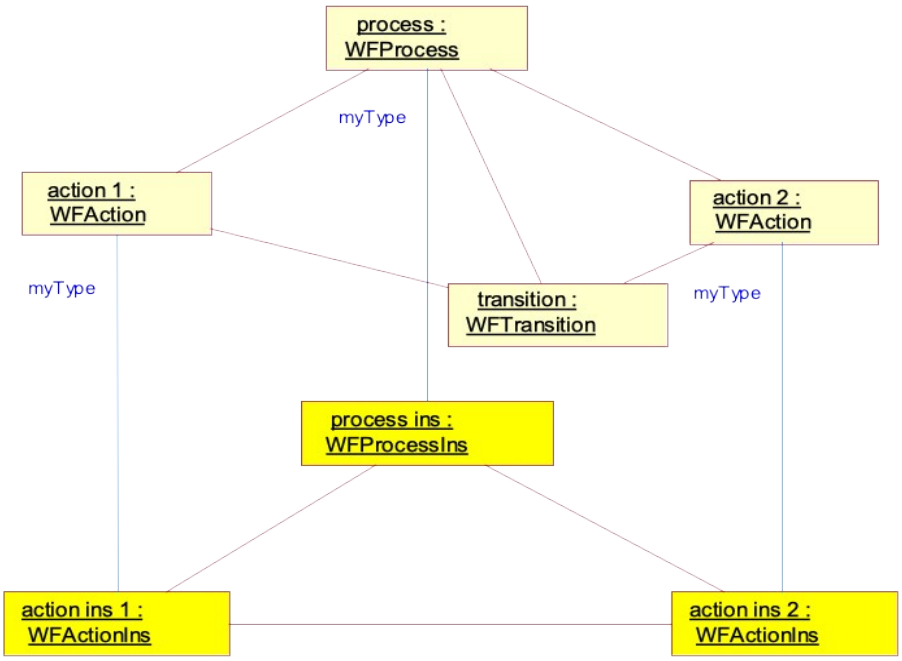


**Figure 9:** Mapping of a Process definition into its implementation

The attribute *state* of the abstraction Activity Instance holds information about the current state of the Activity Instance. The set of possible states is predefined, and it consists of the following states:

- *Created* – The state of an Activity Instance upon its creation, i.e. the initial state of an Activity Instance.
- *Initiated* – The Activity Instance has been initiated, i.e. its resources have been allocated, but the conditions for it to start the execution have not been fulfilled yet.
- *Running* – The Activity Instance has started the execution.
- *Suspended* – The Activity Instance has been temporarily suspended.
- *Completed* – The Activity Instance has fulfilled the conditions for its completion - the work has been successfully completed.
- *Abandoned* – The execution of the Activity Instance has been stopped before it was normally completed.
- *Refused* – The completion of some of the previous Activity Instances was not satisfactory, and the current Activity Instance will not be started until the problem is located and solved.

The transitions from one state to another, as well as the events/commands that trigger them, will be discussed with more detail in the following chapter.

Beside the information about the state, an Activity Instance holds other information, such as the starting time, the completion time, and the deadline by which the work should be completed.

Fork, Join, and Branch Activities also have their implementation in form of Fork Instance, Join Instance, and Branch Instance. As it was explained previously, the execution of these

Activity Instances is simple, and requires no resource data. Therefore, the operations for initial conditions checking would have to be overridden in these classes. Also, due to the special behavior of Branch Instances, the operation for transition condition checking would have to be overridden in the class Branch Instance. The implementation of these operations will be discussed in the following chapter.

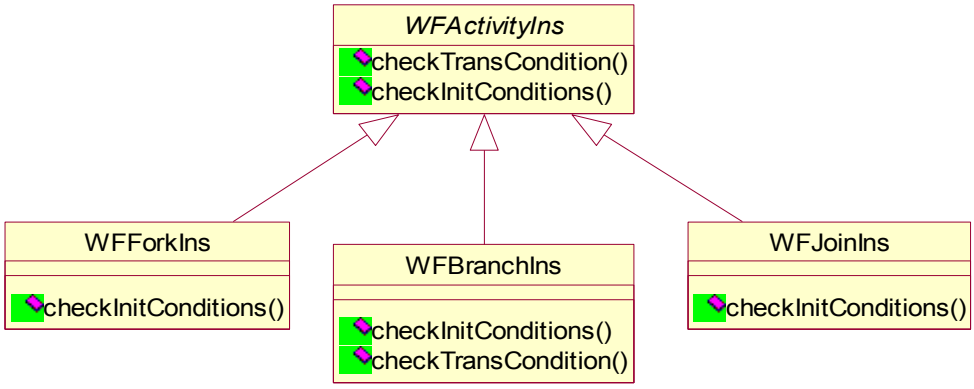


**Figure 10:** Fork, Join, and Brach Activity Instances

*Participants*

The work represented by an Activity Instance may be performed by an arbitrary number of performers, as long as each of the performers plays one of the roles that were specified at design-time.

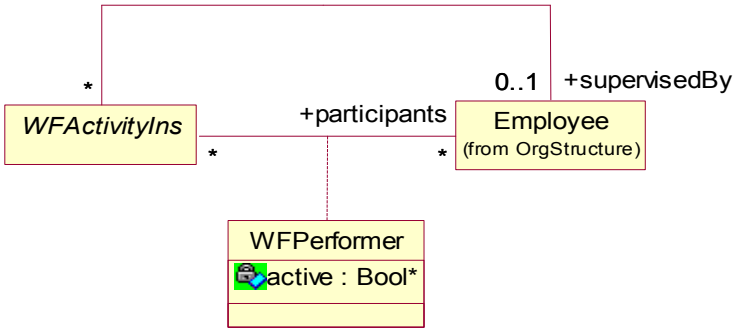


**Figure 11:** Workflow Participants

If a supervisor of an Activity Instance is not specified, then it is implicitly taken that the supervisor of the Activity Instance is the supervisor of its parent Process Instance. It is the supervisor who decides which Activity Instances will be assigned to which participants (Employees). The supervisor may decide to distribute a certain Activity Instance to all the Employees who play the specified role, or only to the selected ones. In either case, some Employees will receive a notification that an Activity Instance has been added to their task list. The one who takes the responsibility for the execution of the Activity Instance is the Employee who starts the Activity Instance. By initiating the Activity Instance, he/she sets the

flag *active* of the *Performer* link-instance. This way, the supervisor will know which of the participants is the actual performer of the Activity Instance's work.

The user that performs the Process instantiation will actually be the supervisor of the created Process Instance.

*Resources*

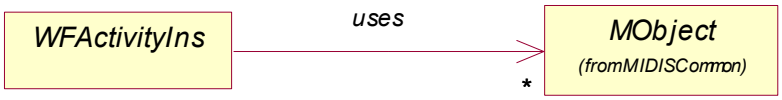


**Figure 12:** Resource usage

Resources that are consumed during the execution of an Activity Instance are actually MObjects. As previously explained, the types of MObjects that are to be used in the execution are specified in the corresponding Resources, associated with the Activity Instance's Activity type. However, the specification of Resources at the definition side is only a recommendation, and not a constraint. An Activity Instance should use MObjects whose types are found in the collection of Resources, but this is not mandatory. An MObject may be either newly created, or an already existing one. As it was mentioned before, the corresponding Resource (if existing) holds information about the properties that are to be modified, as well as about the properties that may only be viewed. If a concrete MObject was assigned to the corresponding Resource object, then the Activity Instance has to use that MObject.

# Behavior

This chapter will discuss the behavioral features of the module Workflow Management. Different use cases will be presented in order to clarify the usage of the module's abstractions and their applicability in a real-world situation. Use cases are divided according to the side on which they are applied. They may be applied either on the definition side (D), or on the implementation side (I). The use cases will be concentrated on the following topics:

- Definition Creation (D)
- Roles Assignment (D)
- Resources Assignment (D)
- Workflow Definition Instantiation (D)
- Participant Assignment (I)
- Initiation and Object Allocation (I)
- Object Usage (I)
- State Transitions (I)

The last section in this chapter will deal with the Workflow Engine, i.e. the automation of state transitions.

## Definition Creation

The procedure of creating a Process definition consists of the following steps:

1. Creating a Process.
2. Creating the Process' sub Activities.
3. Adding the created (or already existing) Activities to the parent Process.
4. Creating the Transitions.
5. Adding the created Transitions to the parent Process.
6. Linking the created Transitions with the corresponding Activities.
7. Specifying the starting Action and the finishing Action(s) (a Process may have only one starting Action and several finishing Actions).
8. Specifying other properties of the created sub Activities, such as their description, duration, etc. (optional).

All of the above stated steps (except the last one) need to be accomplished so that a Process may be instantiated. In case a Process doesn't have a starting Action and at least one finishing Action, it cannot be instantiated. If the user specifies more than one starting Action, he will be warned to redefine the Process.

Another constraint imposed on a Process definition is that each of the Process' sub Activities must have at least one incoming Transition (unless it's a starting Activity) and at least one outgoing Transition (unless it is a finishing Activity). The validity checking is performed

before the Process' instantiation. Its implementation will be explained in the section dealing with the Workflow definition instantiation.

The creation of a Process definition could be realized through the standard UI. However, such a procedure would be highly error prone, and not at all visually comprehensible (especially in cases of more complex Processes). The graphical UI (provided by the MIS module Graph Editor V2.0), on the other hand, remedies all of these weaknesses.

The above explained procedure could be realized through the Graph Editor in the following manner (the numbers in the brackets refer to the corresponding steps in the sequence of steps presented at the beginning of this section):

1. User creates a new Diagram and opens it.
2. User configures the Diagram's Toolbar, so that the Toolbar contains items (buttons) that correspond to the creatable Abstractions of the Workflow Management module. These are, in the first place, the items used for the creation of Processes, Actions and Transitions. Other items may also be found on such a Toolbar (for example, an item for the creation of MSScripts).
3. User selects the Process creation item, and clicks the mouse inside the Graphical View. As a result, a Process is created, and its graphical representation is displayed on the Graphical View (1).
4. User selects the Action creation item, and clicks the mouse inside the created Process' graphical representation. Consequently, an Action is created and linked with the parent Process (2,3). This procedure may be repeated for all new Actions that need to be specified as the Process' sub Activities. In case already existing Actions (or Processes) need to be the sub Activities of the created Process, user may drag them from the browser and drop inside the parent Process' graphical representation. The corresponding links would thus be created.
5. User selects the Transition creation item and clicks the mouse inside the parent Process' graphical representation. A Transition is thus created and linked with the parent Process (4,5). This procedure may be repeated for all the necessary Process' Transitions.
6. User "plugs" the Transitions' graphical symbols with the Activities' graphical symbols, and, as a result, the corresponding links are created (6).
7. User selects the graphical representation of the Action that will be the starting Action in the Process. User double clicks the Action and modifies the value of its *isStart* property. The graphical representation of the Action is changed, so that it will indicate the starting Action. The same could be applied for finishing Action(s) (7).

The Diagram's Toolbar may also contain items for the creation of Fork, Join, and Branch Activities. The procedure for the creation of these Activities is the same as with any Activity, except that a valid Branch Activity must have an associated MSScript.

The next two use cases will explain the additional procedures for defining a complete Process definition.

## Roles Assignment

*Scenario on the user's side*

1. User opens the dialog that displays the list of all possible roles that can be assigned to an Activity.
2. User selects the role and drags it to the Activity (in the graphical mode, the user may drag the role to the Activity's graphical representation).
3. User may also specify the role of the supervisor (in the same way as previously defined).
4. If an Activity is to be performed by an application, then the corresponding Application object would have to be added to the Activity (through a separate dialog).

Upon the completion of previously defined sequence of steps, the role of the future workflow participant, who could be able to perform the corresponding Activity Instance's work, is defined. The role of the Activity Instance's supervisor may also be defined in the similar manner.

## Resources Assignment

*Scenario on the user's side*

1. User opens the dialog that displays the list of all abstractions (they may also be displayed as icons).
2. User selects the desired abstraction's icon and drags it to the Activity (in the graphical mode, user may place the icon into the Activity's graphical representation).
3. A dialog with all the attributes of the selected abstraction is displayed.
4. User selects the attributes that may be modified.
5. User selects the attributes that may only be viewed.

*Implementation*

The implementation of this use case is shown in Figure 13. Upon the completion of the previously defined sequence of steps, the type of an MObject that could be allocated to the Activity's corresponding Activity Instance is fully defined. In the course of workflow execution, the future workflow participant will be able to modify/view only the predefined properties of the allocated MObject.

In this manner, the user would be able to specify the types of MObjects that can be used at run-time, as well as the MObjects' properties that may be modified/viewed. However, it won't be possible to specify more than one properties setting for each of the specified types.

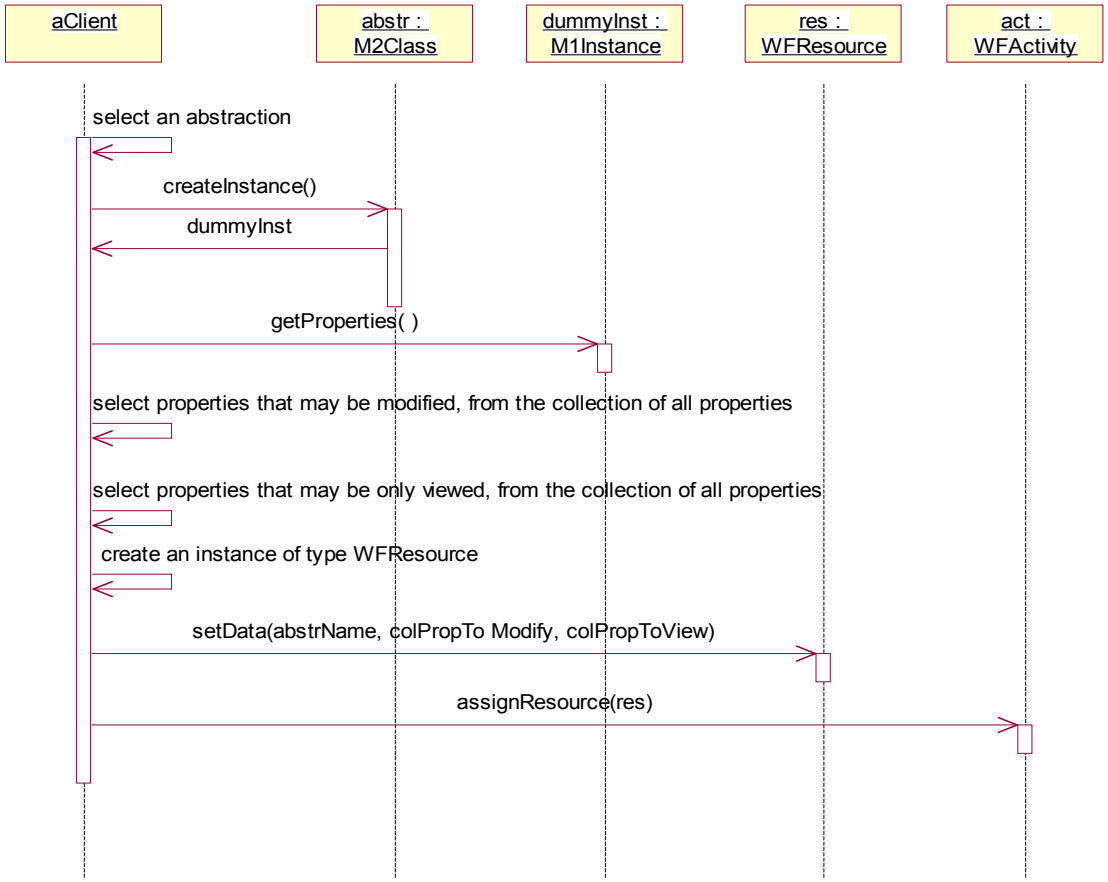


**Figure 13:** Resource Assignment

# Workflow Definition Instantiation

*Scenario on the user's side*

1. User opens a dialog that shows the repository of all created Process definitions.
2. User selects a Process from the repository.
3. User selects the item "Instantiate Process" from the menu, and the command for Process instantiation is applied on the selected Process.

*Implementation*

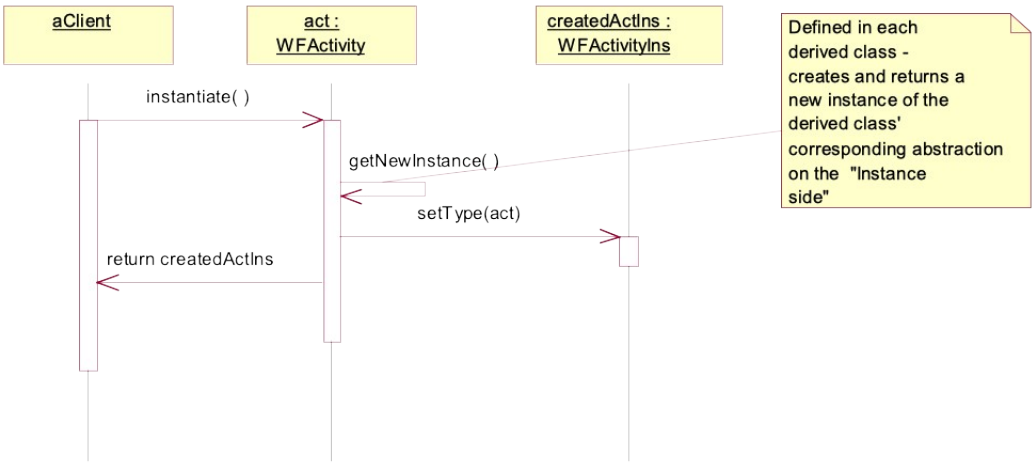


**Figure 14:** WFActivity::instantiate()

During the Process instantiation, the structure of the selected Process' sub Activities and Transitions is traversed and the corresponding Activity Instances are created, along with the links between them. The implementation of this use case is shown in the sequence diagram in Figure 15. The command for Process instantiation calls the operation `instantiate()` of the selected Process. This operation is defined in the class WFActivity (Figure 14) and overridden in the class WFProcess.

Before a Process could be instantiated, the validity of the Process definition would have to be checked. The conditions that have to be satisfied, so that the Process definition is valid, were explained in the section dealing with the definition creation. The implementation of validity checking would require that an operation `checkValidity()` is defined in class WFActivity, and redefined in each of the derived classes. A Process is valid if it has exactly one starting and at least one finishing Action. A Process' sub Activity is valid if it has at least one incoming Transition (unless it is a starting Action) and at least one outgoing Transition (unless it is a finishing Action). A Branch Activity is valid if it has an associated MSScript.

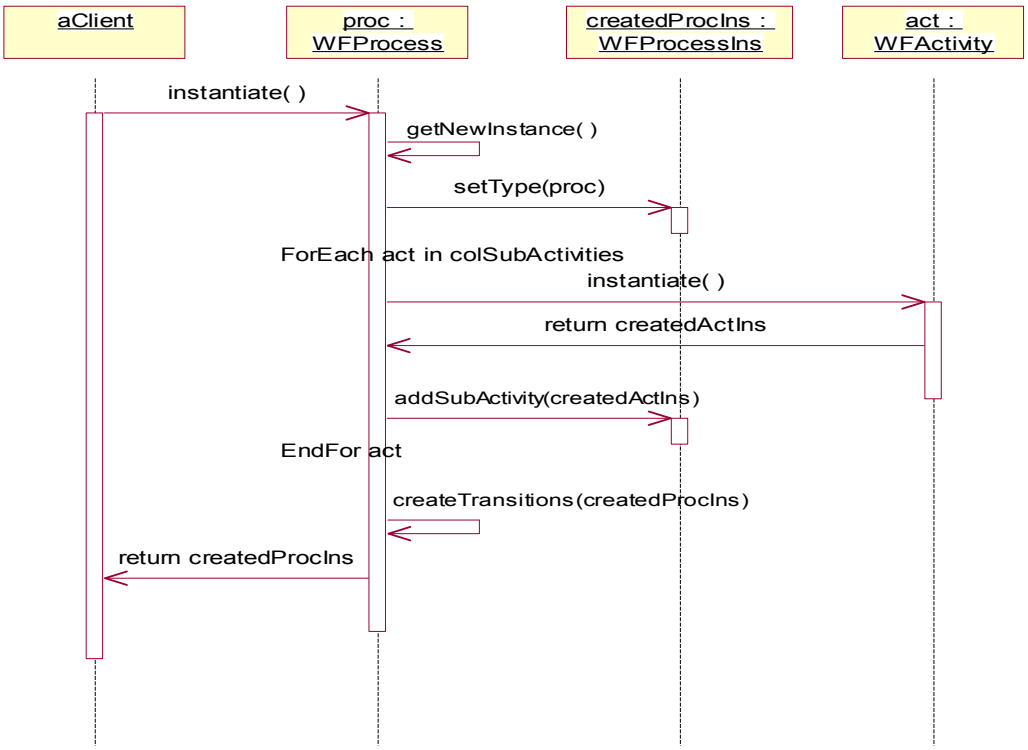


**Figure 15:** WFProcess::instantiate()

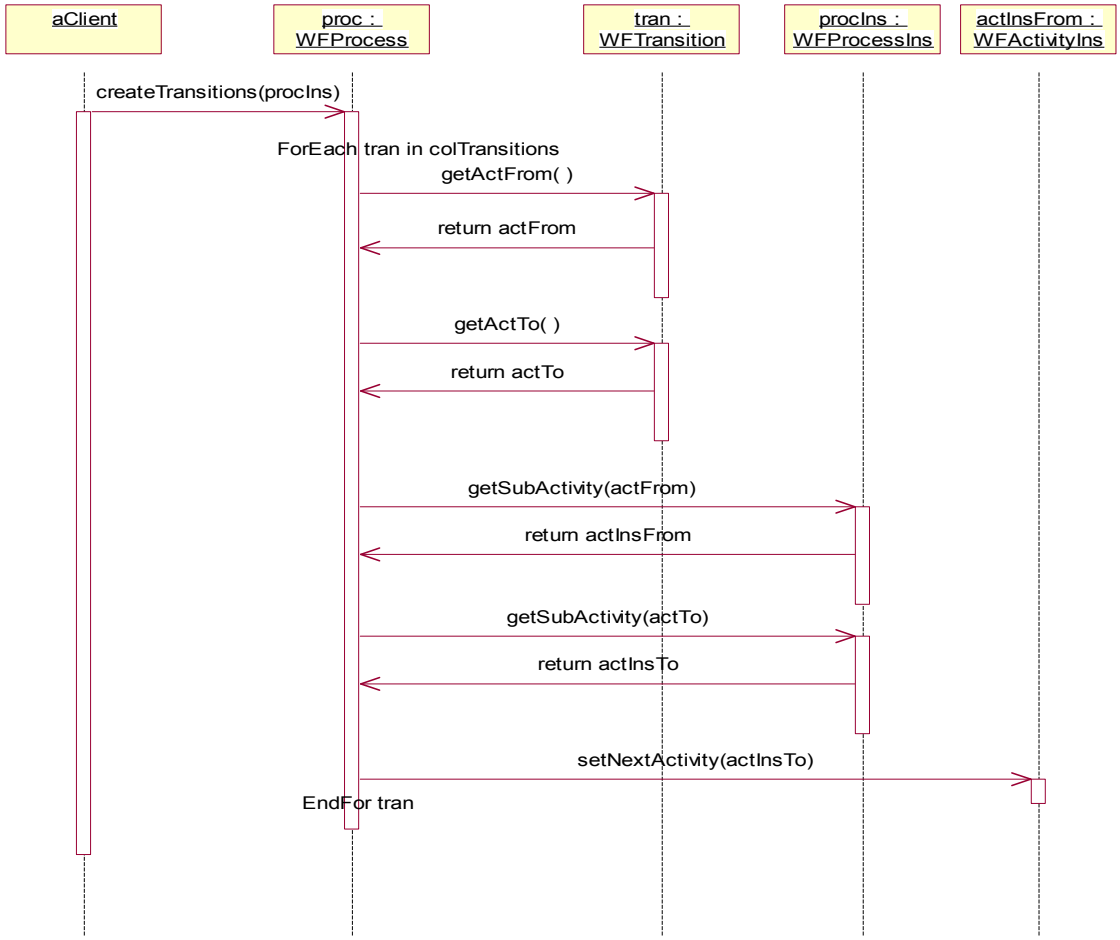


**Figure 16:** WFProcess::createTransitions()

The user that performs the Process instantiation will actually be the supervisor of the created Process Instance. Therefore, he/she must play the role specified by the Process' supervisor Role. The supervisor of the created Process Instance will then distribute the Process Instance to all the possible performers, and the Process Instance will be added to the task lists of the corresponding participants.

When the Process is instantiated, the initial supervisor may also assign supervisors to the created Process Instance's sub Activities. If the supervisors for some of the sub Activities are not explicitly assigned, then it is assumed that they will be supervised by their parent Process Instance's supervisor. Fork, Join, and Branch Activity Instances will have no supervisors (nor performers), since these Activity Instances are performed automatically.

After the Activity Instances have been assigned to them, the supervisors may distribute the Activity Instances to all their potential performers. Activity Instances will then be added to the task lists of the participants.

# Participant Assignment

*Scenario on the user's side*

1. User opens the dialog that displays all the Activity Instances to which he/she is assigned as the supervisor.
2. User selects an Activity Instance and a new dialog is displayed with all the Employees that play the roles defined by the Activity Instance's Activity type.
3. User may choose to assign the Activity Instance to all the displayed Employees, or may select the ones to which the Activity Instance should be assigned.

*Implementation*

The implementation in the case when the supervisor chooses to assign the Activity Instance to all the matching Employees is shown in Figure 17.

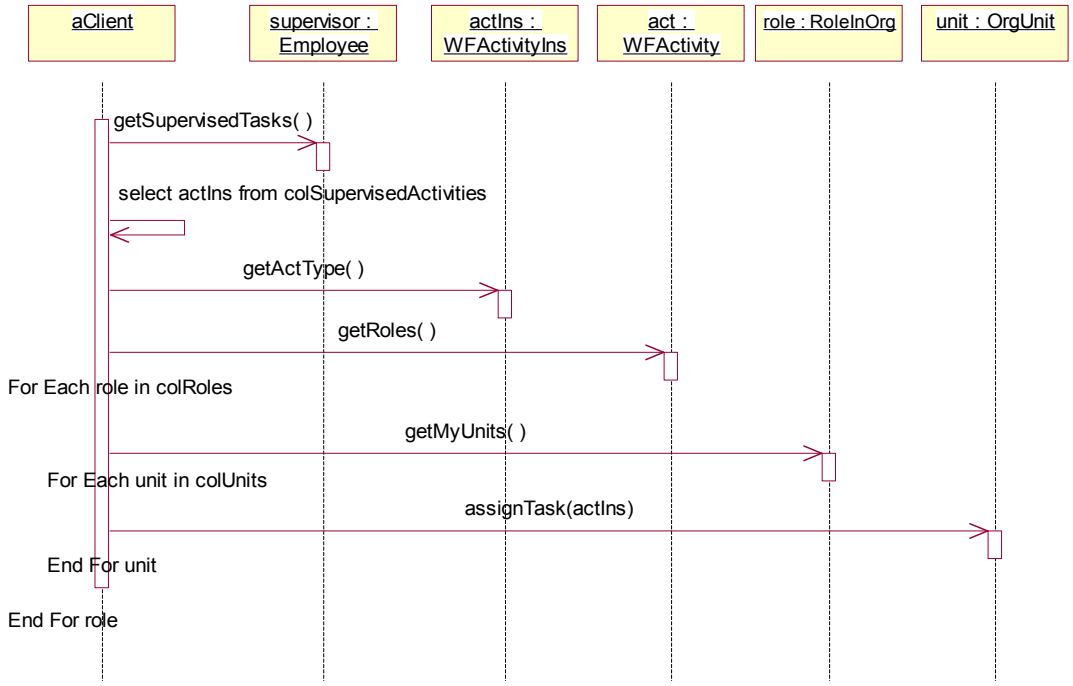


**Figure 17:** Participant Assignment

# Starting an Activity Instance

*Scenario on the user's side*

1. User (a workflow participant) opens the dialog that displays all the Activity Instances assigned to him/her.

2. User selects an Activity Instance and chooses the “Start Activity” item from the menu. By this action, the user takes the responsibility for the execution of the selected Activity Instance. This holds only for the Activities that are not executed by an application.

## Object Allocation

1. User selects an Activity Instance from the list of his currently running Activity Instances.
2. User opens a dialog with all the types of objects needed for the execution of the selected Activity Instance’s work.
3. User may create new MObjects of corresponding types and allocate them to the Activity Instance, or may choose to allocate the already existing MObjects to the Activity Instance. User may also open a dialog with the list of all MObjects allocated to the Activity Instance’s parent Process (or to all of its parent Processes in the Process-subActivities hierarchy). User may select MObjects from the list, to allocate to the Activity Instance.

## Object Usage

During the execution of an Activity Instance, its performer may change the properties of the MObjects allocated to the Activity Instance.

*Scenario on the user’s side*

1. User (the performer) opens the dialog with all the MObjects allocated to the Activity Instance.
2. User double clicks on the desired MObject, and its dialog opens. This may be the standard specification dialog for modification of properties for that object, except that not all of the properties may be modified, and some may not be even displayed. As it was explained earlier, the specification of properties that are to be modified/viewed is given in the Resource that was assigned to the Activity Instance’s Activity type.

*Implementation*

The implementation of this use case is shown in Figure 18.

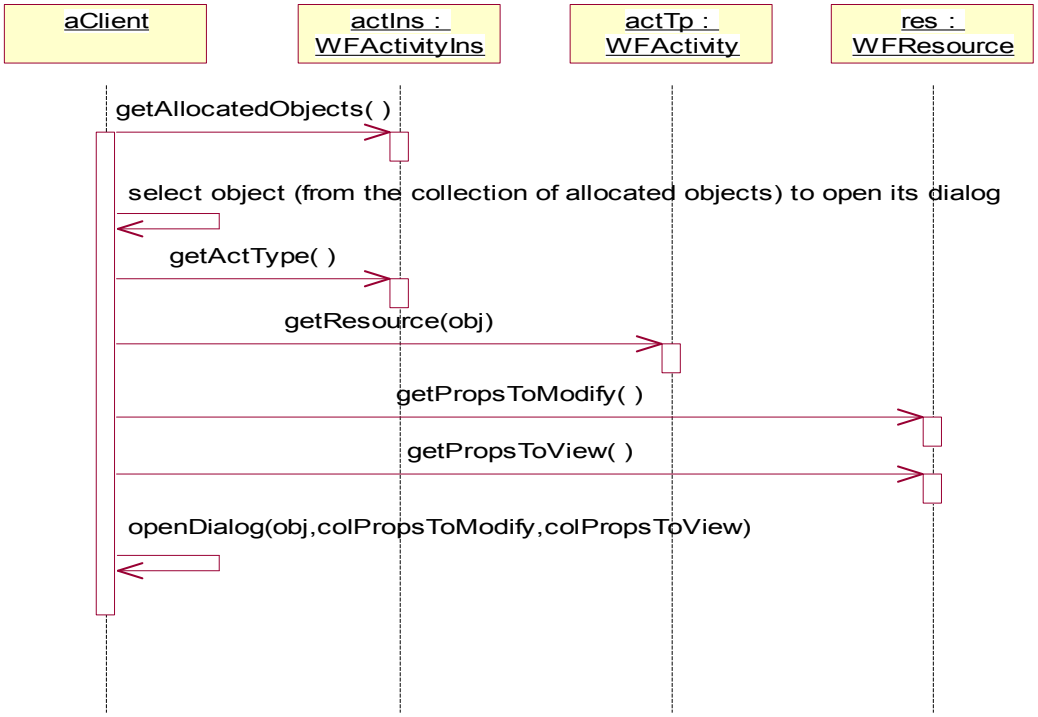


**Figure 18:** Object Usage

# State Transitions

This use case (or, more exactly, a set of use cases) defines the rules for transitions between an Activity Instance's states upon explicit user demand (application of commands for state transitions), or upon completion of an Activity Instance. It also describes some special cases of initiating an Activity Instance.

As it was mentioned in the previous chapter, there is a predefined set of an Activity Instance's possible states. These states are: Created, Initiated, Running, Completed, Suspended, Abandoned, and Refused. Each of these states was briefly explained previously. In this section, a set of strict rules for state transitions will be presented.

Figure 20 shows the state diagram that defines all the possible transitions between the states and the events (commands) that trigger the transitions. The lifecycle of an Activity Instance starts with the state Created and may end with either the state Completed or the state Abandoned. In the first case, the work of an Activity was successfully completed, and in the second, the work was abandoned at some point in the lifecycle of an Activity Instance.

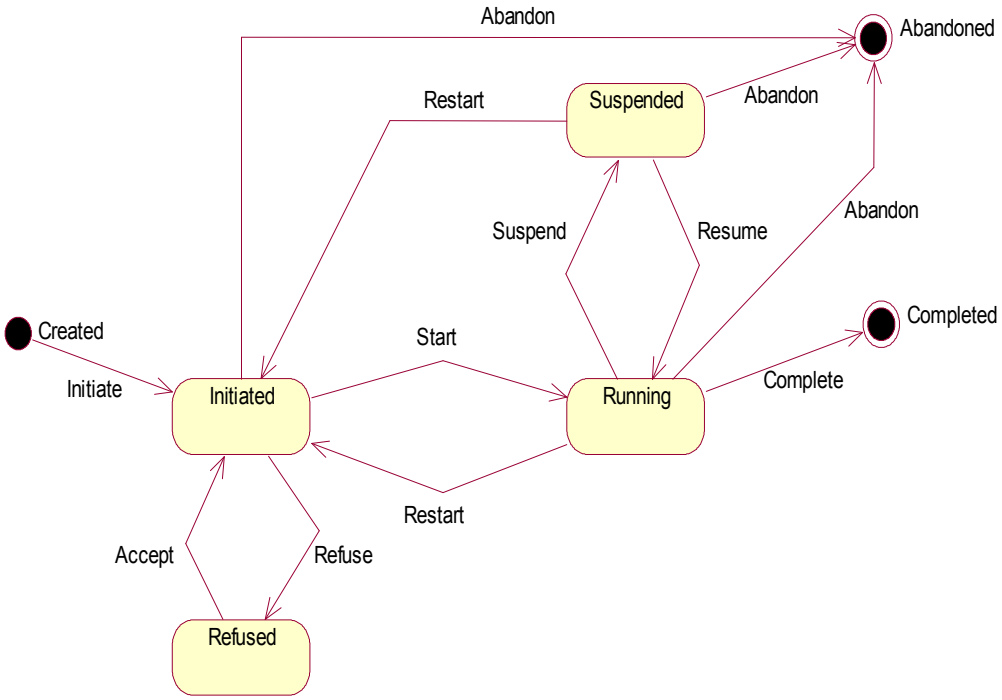


**Figure 19:** State diagram for the lifecycle of an Activity Instance

*Initiation*

When an Activity Instance is created, the initial conditions would have to be satisfied for it to become Initiated. Checking the initial conditions implies checking whether at least one of the

previous Activity Instances is completed.

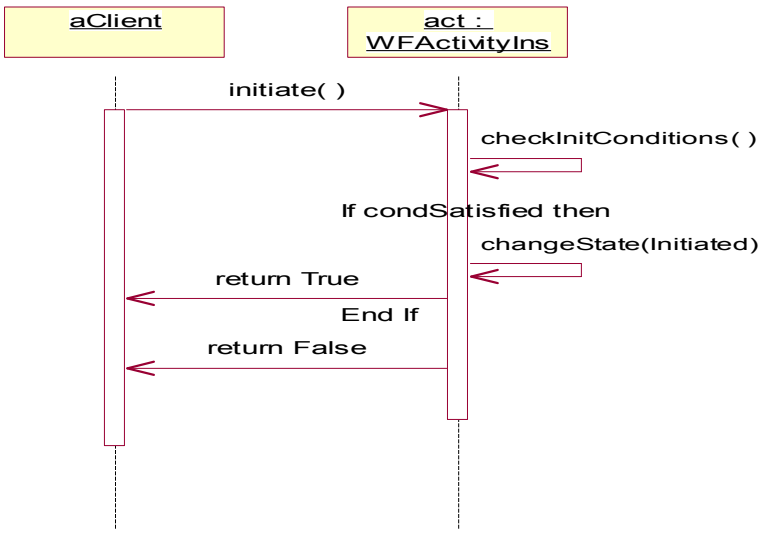


**Figure 20:** WFActivityIns::initiate()

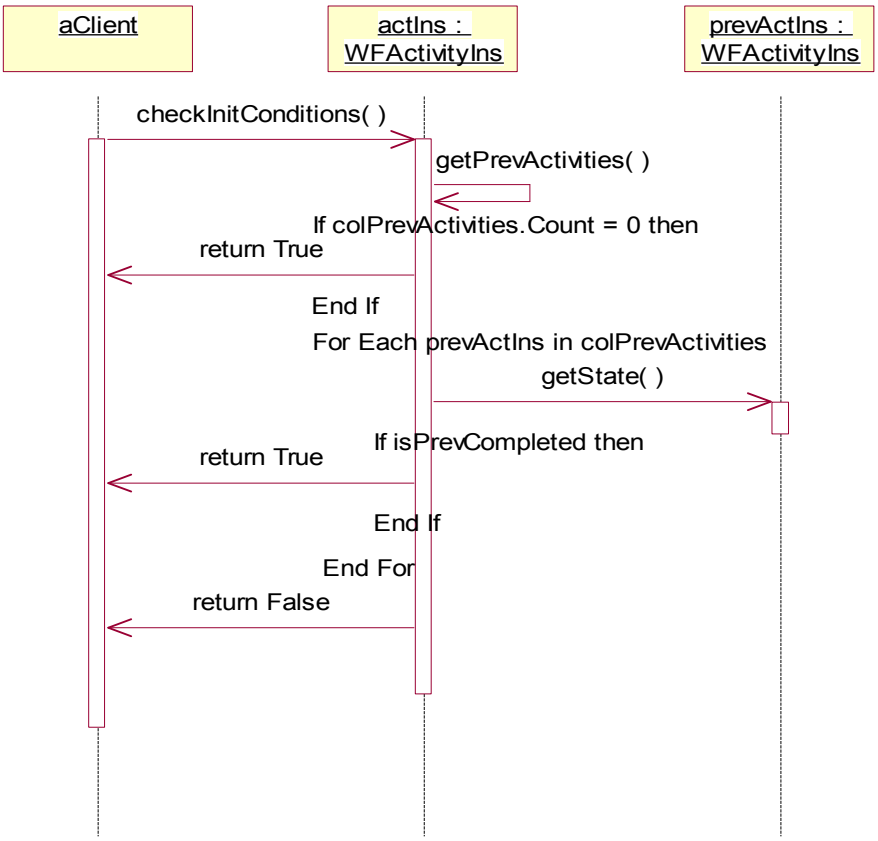


**Figure 21:** WFActivity::checkInitConditions()

In the case of a Process Instance initiation, the Process Instance's starting Action Instance will be automatically initiated (the implementation will be given in the section dealing with the Workflow Engine).The command for initiation may be applied only to a Process Instance. All other initiation (of the parent Process Instance's sub Activity Instances) is performed automatically, in the course of Process Instance execution.

In the case of a Join Activity Instance, a different condition would have to be satisfied for its initiation. Namely, all the Activity Instances that precede the Join Instance would have to be completed, so that the Join Instance could be initiated. For this purpose, the operation `checkInitConditions()` is overridden in the class WFJoinIns.

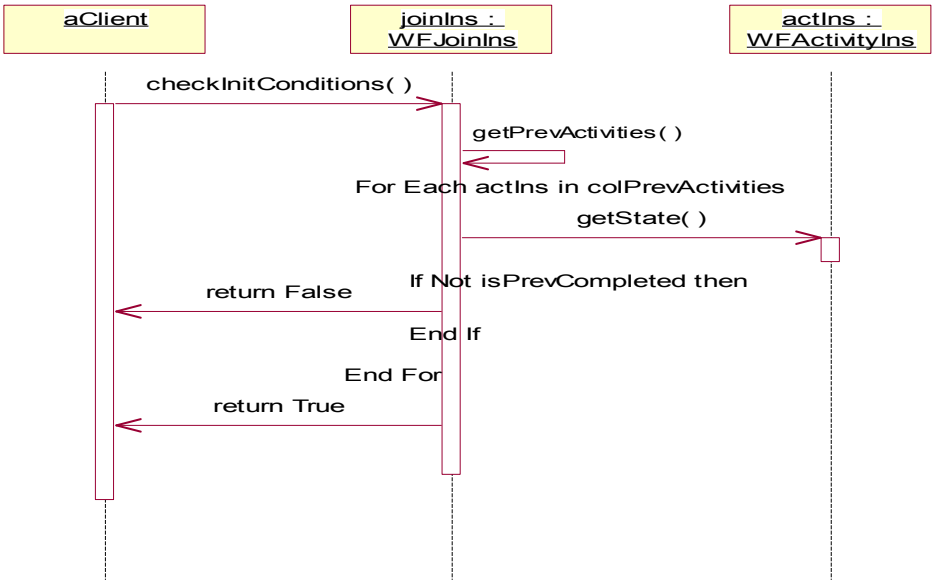


**Figure 22:** WFJoinIns::checkInitConditions()

Fork and Branch Activity Instances have no allocated resources, and, thus, the initial conditions checking is not necessary in these cases. The operation for initial conditions checking is, therefore, overridden in the classes WFForkIns and WFBranchIns, so that it always returns True. However, pre conditions checking (described in the following section) could be performed in these operations.

*Start*

When an Activity Instance is initiated, it may be started on explicit demand of the performer. Before the Activity Instance's state may become Running, the pre conditions would have to be checked.

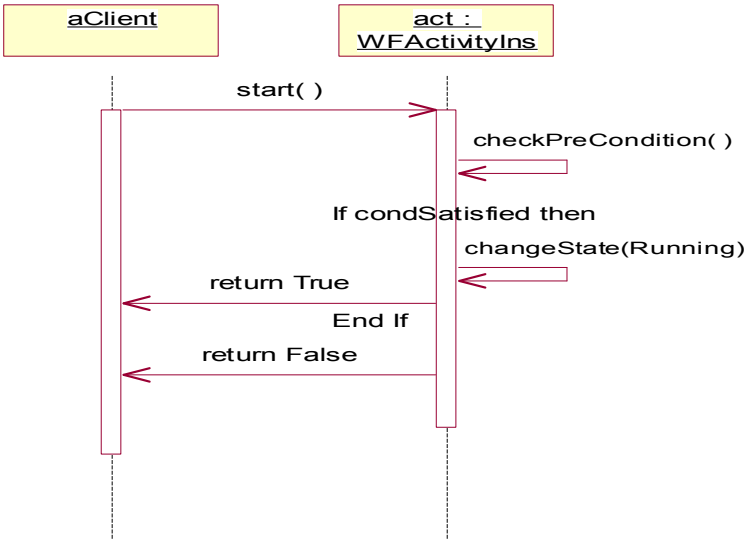


**Figure 23:** WFActivity::start()

A Process Instance cannot be started on explicit demand of the user. A Process Instance is started when the performer of its starting Activity Instance starts the execution of the Activity Instance. However, if the pre conditions of the parent Process Instance are not satisfied, then neither the Process Instance, nor its starting Activity Instance can be started. The implementation is given in Figure 24.

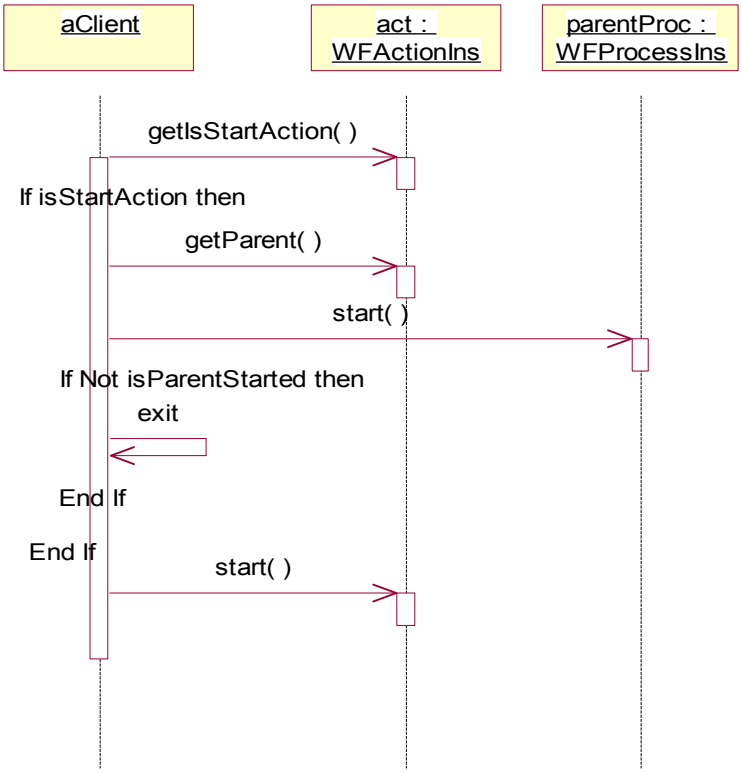


**Figure 24:** Start Action

*Complete*

When the performer completes the execution of an Activity Instance, he/she explicitly applies the command for completion on the Activity Instance. The Activity Instance may change its state to Completed, only if the post conditions are satisfied.

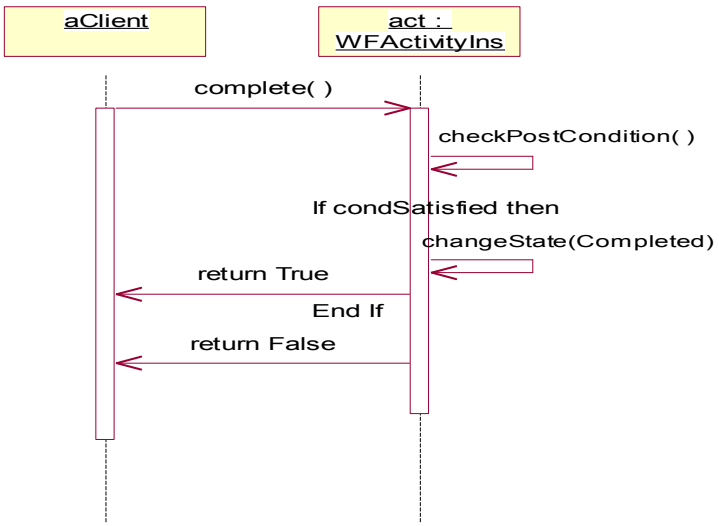


**Figure 25:** WFActivity::complete()

When a Process Instance's finishing Action Instance is completed, the Process Instance may also be completed if its post conditions are satisfied. In case they are not satisfied, the explicit command for Process Instance completion would have to be applied later.

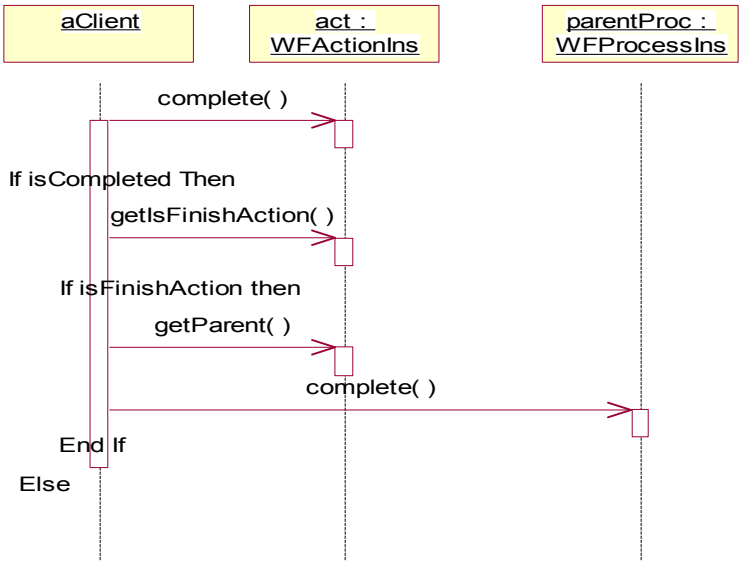


**Figure 26:** Complete Action

In case a completion command is applied on the Process Instance, the Process Instance may change its state to Completed only if at least one of its finishing Activity Instances is completed.

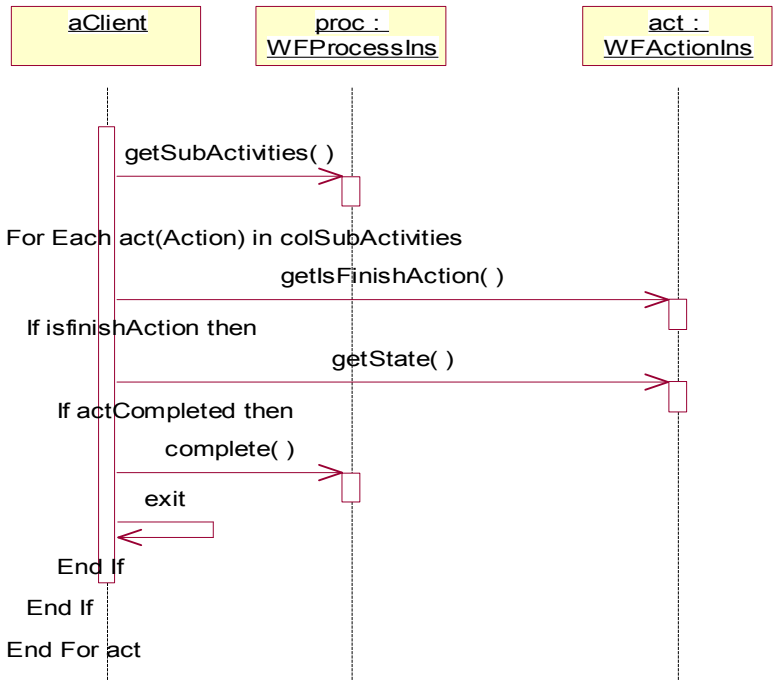


**Figure 27:** Complete Process

When an Activity Instance is completed, the following Activity Instances will be automatically initiated. The implementation of this automation will be given later in the text.

*Suspend*

In the course of an Activity Instance's execution, its performer may decide to temporarily suspend the execution. He/she will, therefore, apply the command for Activity Instance suspension.

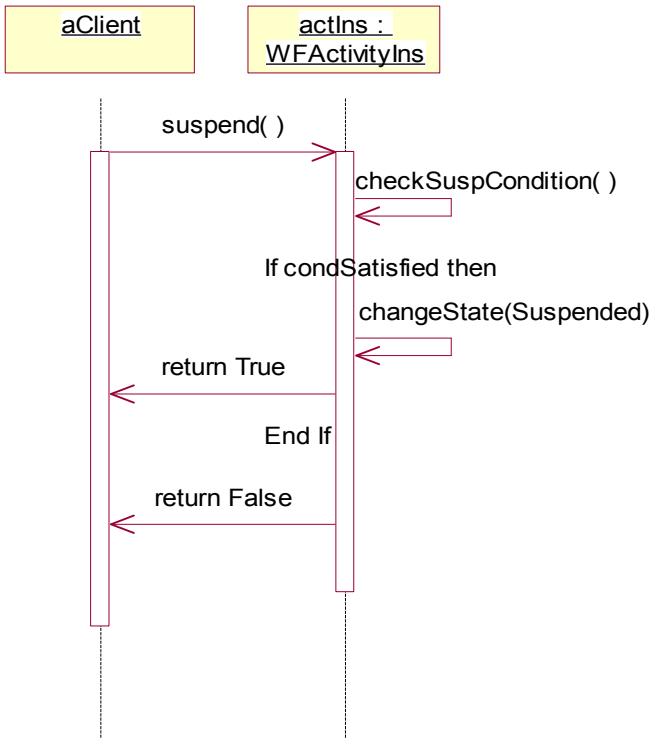


**Figure 28:** WFActivityIns::suspend()

In case of a Process Instance suspension, all of the Process Instance's sub Activities would have to be notified that their parent Process Instance is suspended. They will be notified by setting the flag *isParentSuspended* of class WFActivityIns. That way, any further operations on the Process Instance's sub Activities will be forbidden until the execution of the parent Process Instance's execution is resumed. The setting of the flag will be performed recursively.

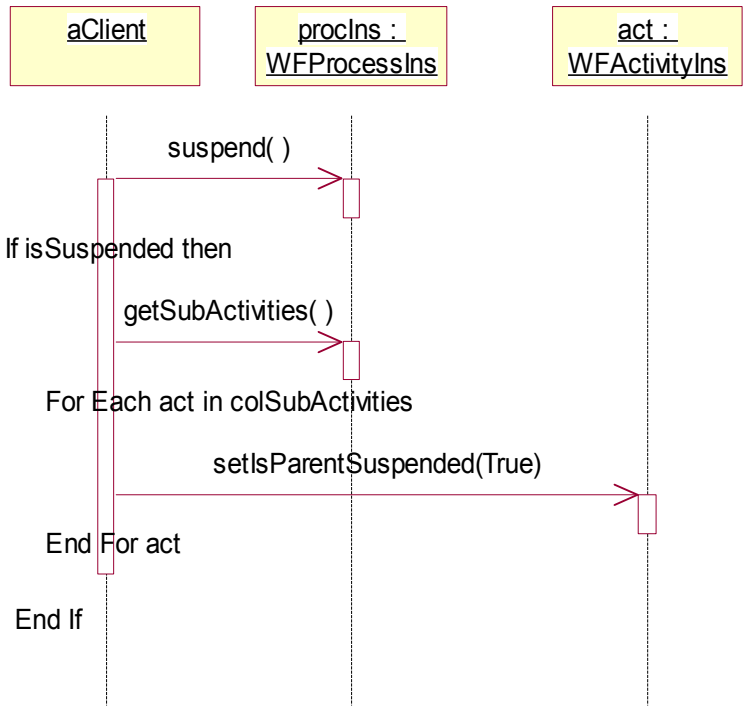


**Figure 29:** Suspend Process

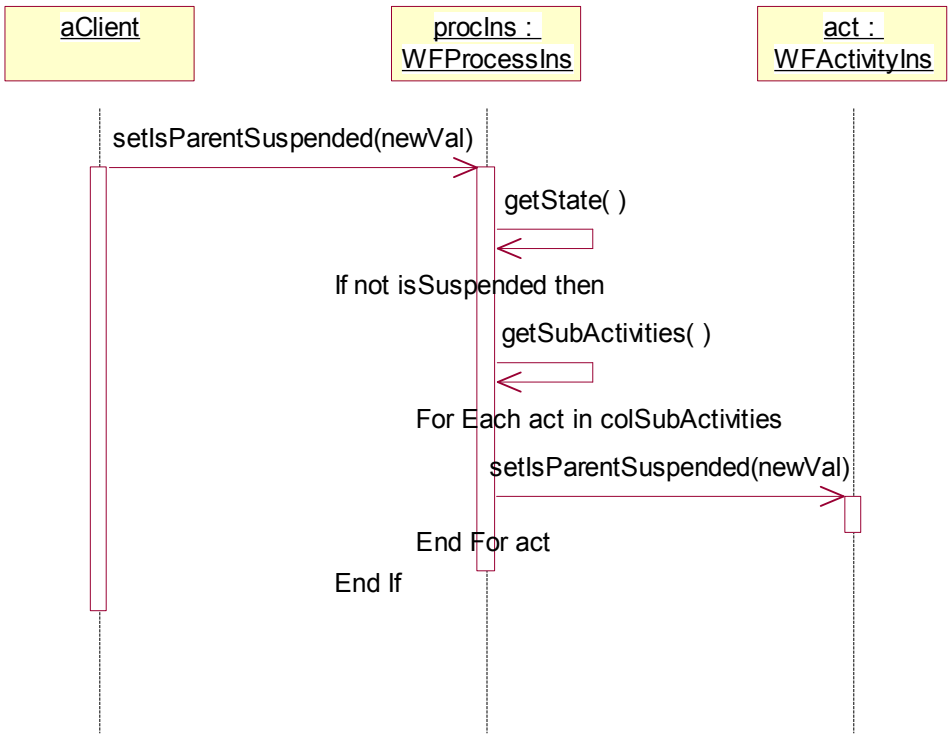


**Figure 30:** WFProcessIns::setIsParentSuspended()

*Resume*

If an Activity Instance is in the Suspended state, the performer may choose at some point to resume the execution of the Activity Instance.

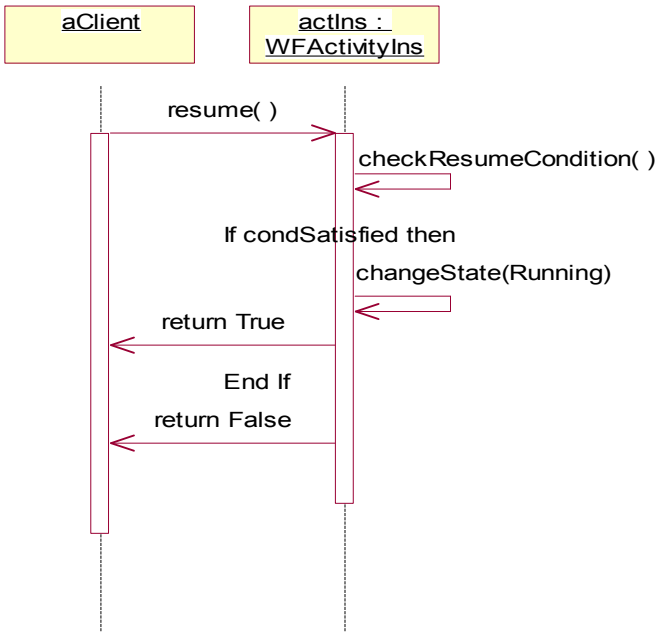


**Figure 31:** WFActivityIns::resume()

In case a Process Instance is resumed, all of its sub Activities would have to be notified that the Process Instance is again in the Running state, so that the normal execution of these Activity Instances can be resumed. For this purpose, the *isParentSuspended* flag is reseted.

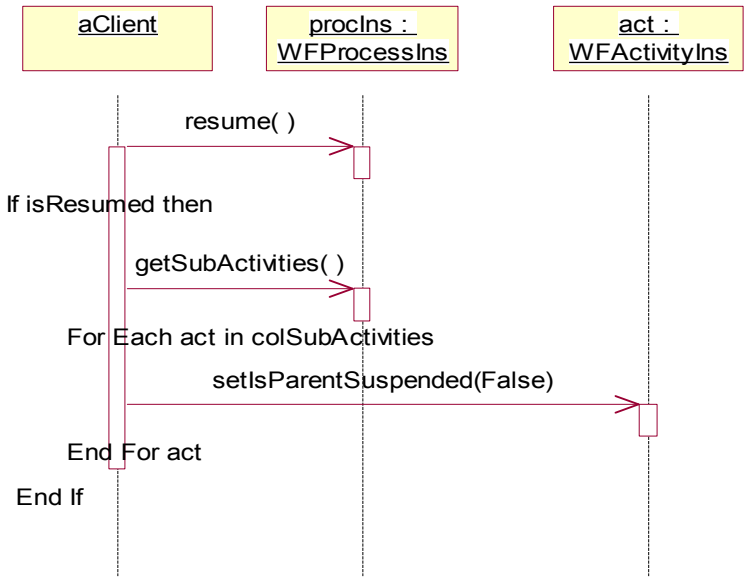


**Figure 32:** Resume Process

*Abandon*

The execution of an Activity Instance may be abandoned at any time in the lifecycle of the Activity Instance.

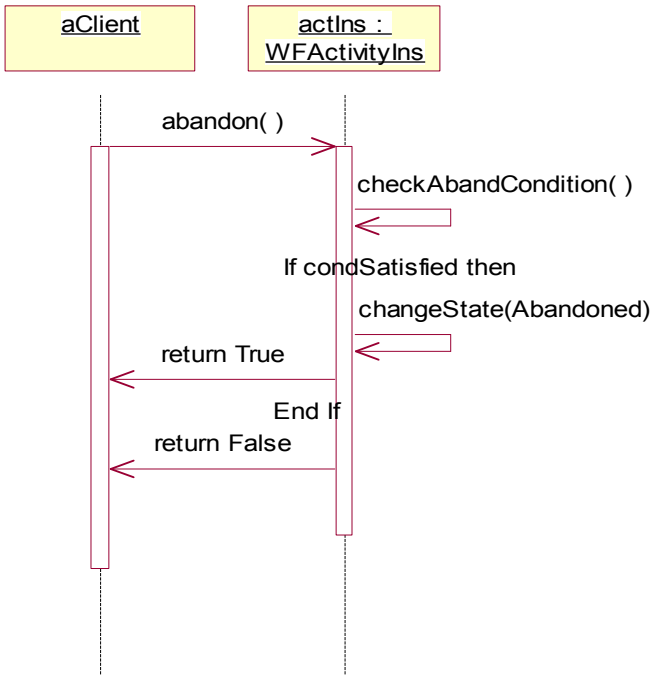


**Figure 33:** WFActivityIns::abandon()

In case of abandoning a process Instance, all of its sub Activities would have to be abandoned, too. The abandoning is performed recursively.

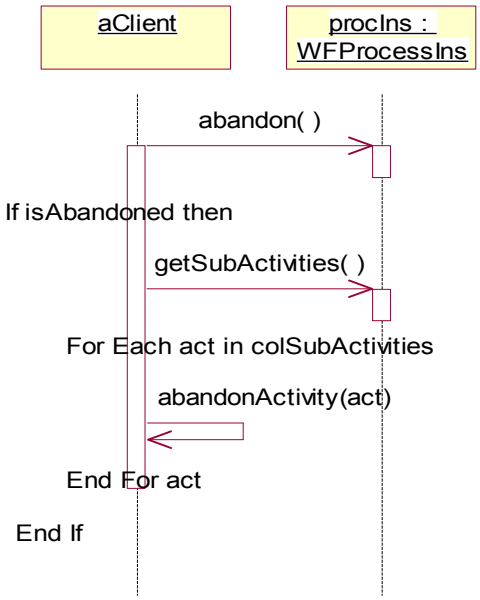


**Figure 34:** Abandon Process

# Workflow Engine

Not all of the state transitions will be triggered on explicit demand of the performer. Some of the state transitions may be automatically triggered on the occurrence of certain events. These state transitions will be discussed in this section. The events that may trigger the transitions are specific commands applied on Activity Instances.

For the purpose of state transitions automation, the singleton class *WFEngine* is introduced. Its purpose is to provide the automation functionality. It may also act as a repository of all created Process definitions, so that each time a Process is created, it may be linked to the Engine.

When a command is applied on an Activity Instance, an internal Event is generated, and Engine acts as an Event client. The Engine may interpret the Event, and react accordingly upon it.

The Engine abstraction provides the automation for the following cases:

- When a Process Instance is initiated, its starting Activity Instance is initiated as well
- When Fork, Join, or Branch Instances are initiated, the command for their completion should be immediately applied.
- When an Activity Instance is completed, its outgoing transitions should be fired, and the corresponding Activity Instances should be initiated.

*On initiating a Process Instance*

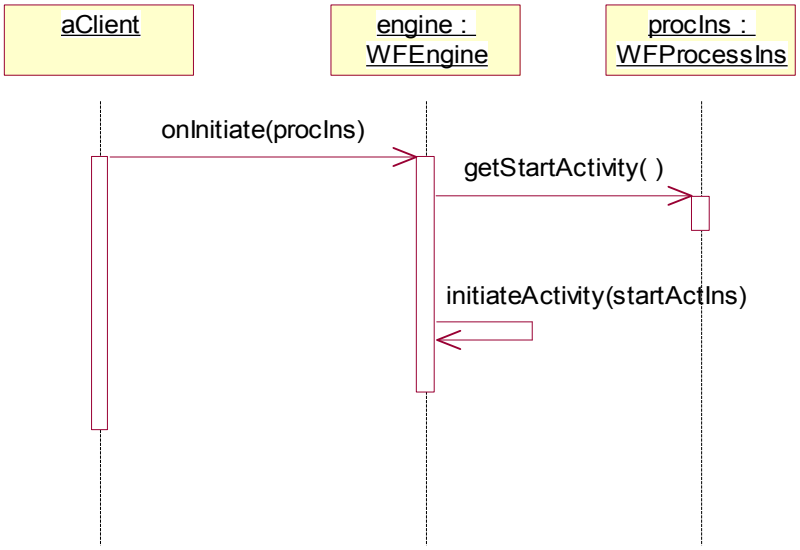


**Figure 35:** WFEngine::onInitiate(procIns)

When the command for initiation is applied to an Activity Instance, and the command is successfully executed, the corresponding Event is generated, and the Engine is notified of the Activity Instance's initiation. The case when the initiated Activity Instance is actually a Process Instance, the starting Activity Instance is initiated as well. The implementation is shown in Figure 35.

*On Initiating a Fork, Join, or Branch Activity Instance*

The case when the initiated Activity Instance is actually a Fork Instance is shown in Figure 36. The sequence diagram is the same for Join and Branch Activities.

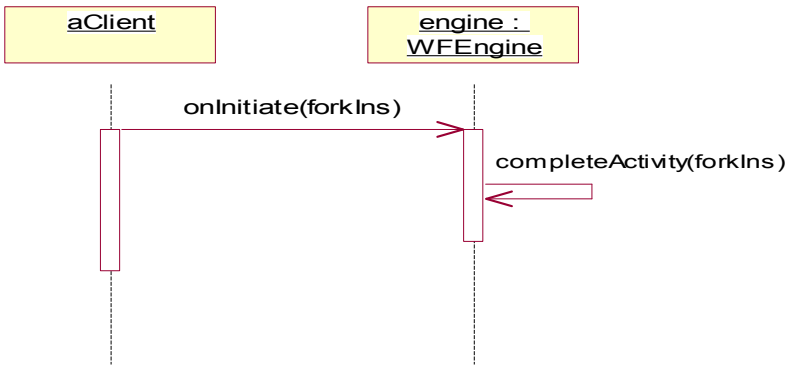


**Figure 36:** WFEngine::onInitiate(forkIns)

*On completing an Activity Instance*

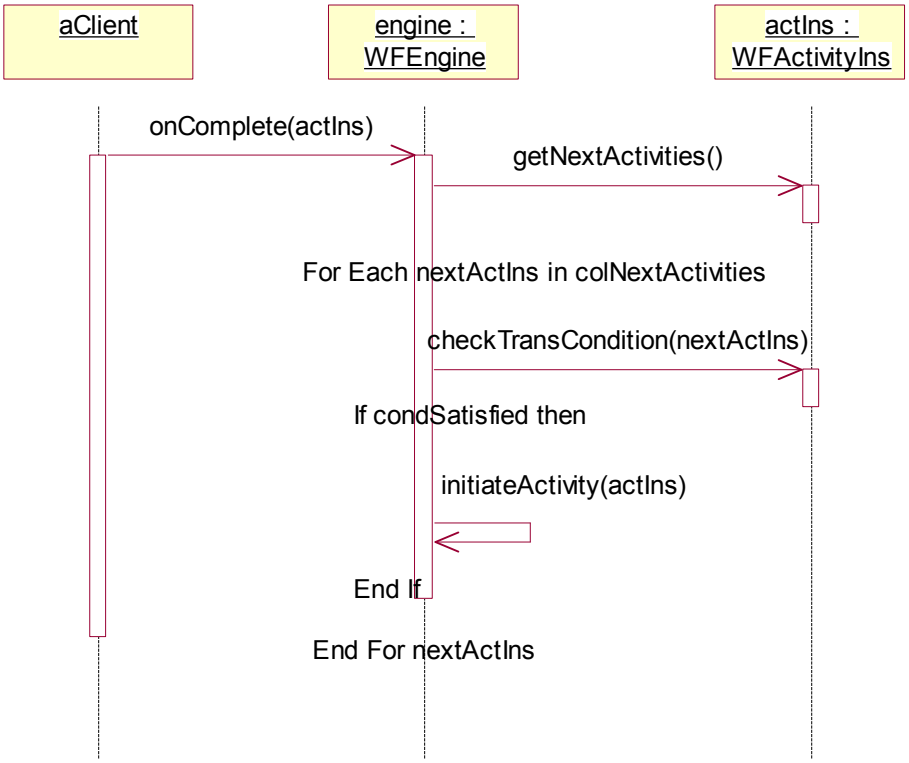


**Figure 37:** WFEngine::onComplete()

When the command for completion is applied to an Activity Instance, and the command is successfully executed, the corresponding Event is generated, and the Engine is notified of the Activity Instance's completion. All the Activity Instances that follow the completed one are inspected for the transition conditions. If the transition conditions are satisfied, the Activity Instances are initiated. The implementation is shown in Figure 37.

The implementation of operations for transition conditions checking are shown in Figures 38 and 39.

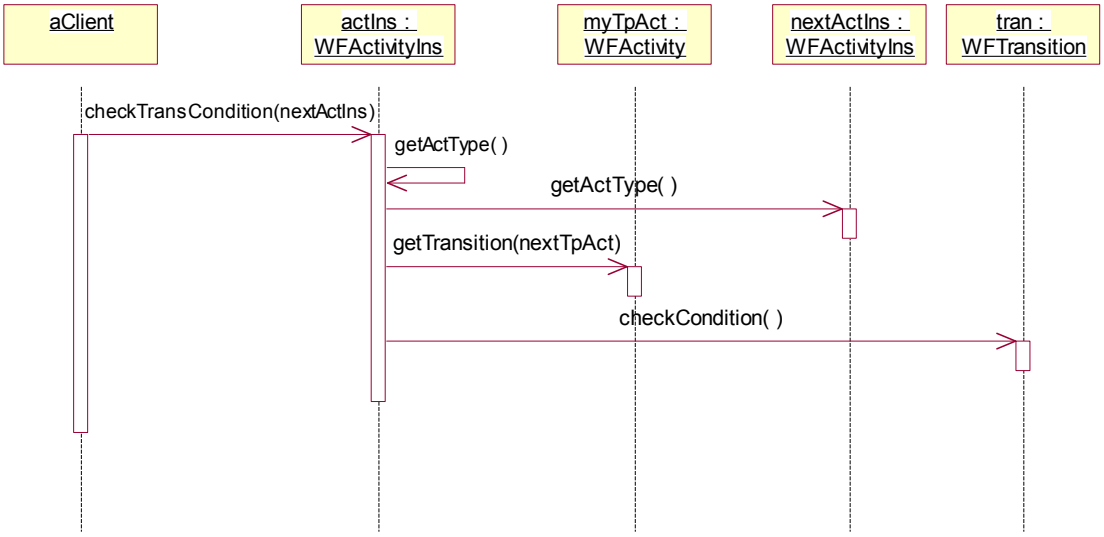


**Figure 38:** WFActivityIns::checkTransCondition()

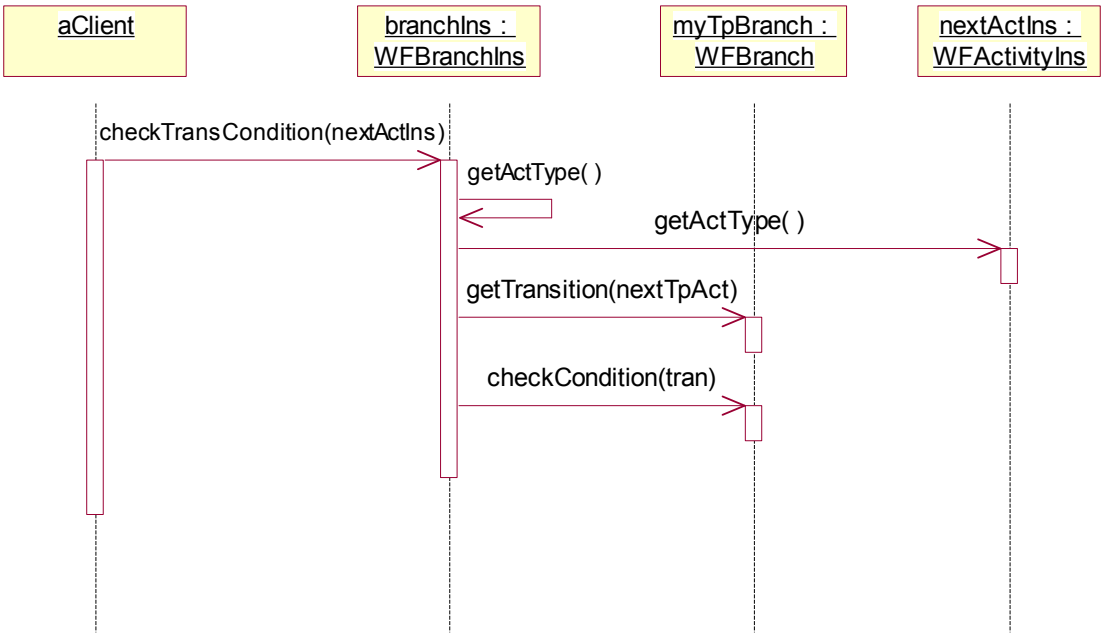


**Figure 39:** WFBranchIns::checkTransCondition()

# Addendum

## Overview of Provided Functionality

This chapter will give a summary of the services that are offered to the user, which may either act as the creator of a Process definition or the supervisor/performer of the Process implementation.

When acting as a Process definition creator, beside the Graph Editor's main graphical area (in case the graphical UI is used), the user will be able to view the following dialogs.

- The dialog that displays all the Roles that may be assigned to an Activity. The user may select a Role and assign it to the created Activity.
- The dialog that displays all the MObject types that can be assigned to an Activity. The user may select the MObject types and specify these types as the Activity's Resources.
- The dialog that shows all the Attributes of the selected abstraction. The user may select the properties that are to be modified, as well as the ones that may only be viewed.
- The dialog that displays all the Applications that may be assigned to an Activity. Assigning an Application to an Activity would mean that the Activity's execution would be controlled by the specified application.
- The dialog that displays the longest path of the Process execution. This may be a separate dialog that displays the maximum time of Process execution, as well as the Activities belonging to the path. The path may also be indicated on the Process definition's graphical representation, by graphically accentuating the Activities that belong to the path, i.e. by changing a graphical property of these Activities.
- The dialog that displays the shortest path of the Process execution. When calculating the longest/shortest path, only the straightforward Process execution is considered, and the loops are ignored.
- The dialog that shows all the created Process definitions. The user may select a Process, and choose to instantiate it. He/she will thus become the supervisor of the created Process Instance.

When acting as a Process Instance's supervisor, the user may view the following dialogs.

- The dialog that displays all the Process Instances for which the user is the supervisor.
- The dialog that displays all the Activity Instances of the selected Process Instance. In case of graphical operational mode, this does not have to be a separate dialog.
- For each of the Activity Instances, the user may view the dialog that displays all the Roles that the supervisor of the Activity Instance may play. For each of the roles, the user may view the dialog that shows all the Employees playing that role in the organisation. The user may select Employees from the dialog and assign them as supervisors to each of the Activity Instances.

When acting as an Activity Instance's supervisor, the user may view the following dialogs.

- The dialog with all the Activity Instances for which the user is the supervisor.
- The dialog that displays all the roles that the potential performers of the selected Activity Instance may play. For each of the Roles, the user may view the dialog that shows all the Employees playing that Role in the organisation. The user may select the Employees from the dialog and distribute the Activity Instance to them.

When acting as a potential performer of Activity Instances, the user may view the following dialog.

- The dialog that displays all the Activity Instances that are assigned to the user. The user may select an Activity Instance and choose to start its execution. He/she will then declare him/herself as the actual performer of the Activity Instance.

When acting as an actual performer of Activity Instances, the user may view the following dialogs.

- The dialog that displays all the Activity Instances that the user performs.
- The dialog that displays the selected Activity Instance's Resources, i.e. the types of MObjects that should be allocated to the Activity Instance. This dialog serves only as a reminder of the types of MObjects that need to be allocated. It is not obliging in any way.
- The dialog that displays all the MObjects allocated to the Activity Instance's parent Process Instance(s). The user may select these MObjects and allocate them to the Activity Instance.
- The dialog that displays all the MObjects that were allocated to the Activity Instance. The user may select an MObject and open its dialog (specification).

When acting as a supervisor/performer of a Process Instance, the user will be offered a possibility to view the current state of the Process Instance at any time during its execution. The following features may define the current state of the Process Instance:

- The Activities that are currently in the Running state, and their performers.
- The Activities that are late with their execution, i.e. their deadline time has passed.
- The Activities that are not yet completed, but the deadline for their execution is near.
- The expected time(s), by which the Process Instance should be completed. If there is more than one path that leads to the completion of the Process Instance execution, then the expected time is calculated for each of the paths, from the expected execution times of Activity Instances that are found on the paths.
- The maximum time by which the Process Instance will be completed.
- The minimum time by which the Process Instance may be completed.

# Enhancements

This section will give a brief summary of the features of the MIS module Workflow Management V2.0 that are not supported by the MIS module Workflow Management V1.0. These are the following.

- Roles for the workflow participants may be specified at design-time.
- Roles for the supervisors may be specified at design-time.
- Fork, Join, and Branch Activities may be defined.
- Work of an Activity may be performed by a computer application.
- Types of MObjects that may be used as Resources may be specified at design-time, as well as the property settings of each type of MObjects.
- There is a possibility to specify the minimal, maximal and expected time of each Action within a Process. There is also a possibility to perform various calculations based on these time values. These calculations are useful for finding the shortest/longest path of the Process execution.
- The supervisor may be assigned to an Activity Instance.
- The state Refused is added to the set of an Activity Instance's possible states.