Scheduling engine, resource editor and orchestration module of FlowOpt project

System Requirements Specification

Version: 0.2

Date: 2010/23/03

Version history

Date	Version	Author	Description
2010/10/03 2010/16/03	**0.1**	**Milan Jaška**	**Document structure**
2010/23/03	**0.2**	**Milan Jaška**	**Basic format descriptions and semantics**

This document is part of FlowOpt project - student software project at Faculty of Mathematics and Physics of Charles University in Prague.

1 Table of contents

2 Introduction

2.1 Content and purpose

This document specifies requirements on the scheduling engine, resource editor and orchestration modules of the FlowOpt project.

2.2 FlowOpt project

FlowOpt project is a student software project at Faculty of Mathematics and Physics of Charles University in Prague.

Project aims are to create modular software for editing workflows and resources, creating a scheduler tasks from them, automated scheduling and viewing the schedule in the form of Gantt Charts.

2.3 Modules

The idea of modules is to create a set of applications that may be executed and used separately but which share some interfaces and file formats so that output of one application can be used as input of another application.

The list of modules:

1) Workflow editor
2) Resource editor
3) Scheduling engine
4) Schedule visualize
5) Schedule analyzer
6) Orchestration module

Orchestration module is supposed to be the application that helps user to integrate all those modules to one application and to offer user "Projects" creation. Projects may contain arbitrary number of workflows, resources, scheduling tasks, schedules and analysis as well as information about their connections (for example: "this schedule is a generated for that scheduling task and this scheduling task is a result of usage that workflows and that resources" etc.).

3 Workflows and resources to Scheduling task

Scheduling task is an input for Scheduling engine. Output of scheduling engine is a schedule or information that schedule cannot be generated under the given conditions.

Workflows and resources are formally described in declarative format.

3.1 Declarative format of Workflow and resources description

Workflow and resources are described by their declarative transcription that uses predicates described in this chapter. Predicate order counts.

3.1.1 Terms used in predicates

Terms are case sensitive strings matching a regular expression $[a-zA-Z][a-zA-Z0-9\_]^*$.

inf is a special term for usage in predicates where it stands for an "infinite number". Another special term is ***any***, which is used for "unknown value". This term may be useful in composite activity declaration where the duration of the activity may be unknown till the time the schedule is generated.

3.1.2 Predicate "Activity"

This predicate is used for declaration of activity (or task) that is a part of scheduled process.

$$Activity(A, D_U, C, D_D, C_E, C_L)$$

This declaration says: There exists activity *A*. This activity takes the time of D_U and costs amount of money *C*. This activity should finish not later than at the time of D_D. There are also parameters for costs when the activity is ended earlier (C_E) or later (C_L) than at a time of D_D. The real expense *E* spent on the activity which was finished at a time of *T* is calculated like that:

$$E = C + (D_D \dot{-} T) * C_E + (T \dot{-} D_D) * C_L$$

Operator $\dot{-}$ is special subtraction operator that results in value not less than value of zero.

3.1.3 Predicate "Decomposition"

This predicate is used for declaration of an activity being a composite activity. This predicate also says which activities the activity contains and what are the conditions for their execution.

$$Decomposition(A, [A_1, \dots, A_n], T)$$

This declaration says: The activity *A* is a composite activity that contain activities $A_1, \dots, A_n$. There exist two types *T* of decomposition – *AND* and *XOR*. *AND* decomposition means that when activity *A* is executed then all activities it contains are executed. *XOR* decomposition means that exactly one of activities activity *A* contains is executed.

In other words, decomposition allows us to declare activity being one the following types:

1) Bigger activity (like assembling a car) that contains smaller sub-activities (like mounting a wheel, mirrors, etc.)
2) Alternatives (coming home by front door or back door)

Each of activities A_1, …, A_n may also be composite and all of those activities A, A_1, …, A_n must be declared with "Activity" predicate before used in predicate "Decomposition". Each sub-activity may appear only once as a sub-activity of some activity. It's impossible to declare one sub-activity being sub-activity of two (or more) different composite activities.

3.1.4 Predicate "Temporal"

This predicate is used for declaration of (binary) temporal conditions between activities.

$$Temporal(A_1, A_2, min, max, T)$$

This declaration says: There exists temporal condition between activities A_1 and A_2. This condition may be of the following types:

Type	Explanation
ES	End-to-start
SS	Start-to-start
EE	End-to-end
SE	Start-to-end

Tabulka 1 - Types of temporal conditions

Probably the most useful type of temporal condition is *ES*. This type says that between finishing the execution of activity A_1 and starting the execution of activity A_2 must be at least *min* and up to *max* amount of time.

There are some semantic limitations for usage of temporal conditions. None of sub-activities of a composite activity may be required to start before or end after the activity itself. The other limitation is for circular dependencies – if for example activity A_2 is required to start after A_1 is finished and A_1 is required to start after A_2 is finished. Those limitations have very natural causes.

3.1.5 Predicate "Logical"

This predicate is used for declaration of (binary) logical conditions between activities.

$$Logical(A_1, A_2, T)$$

This declaration says: There exists logical condition between activities A_1 and A_2 and this condition uses operator *T*. Operator may be logical implication ("=>"), equivalence ("<=>"), disjunction ("or") or exclusive disjunction ("xor").

3.1.6 Predicate "Attribute"

This predicate is used for declaration of "attributes" of resources which state for the resources qualities, abilities, etc.

$$Attribute(A)$$

This declaration says: There exists attribute *A*. This attribute may be used for resource characterization and activity resource requirements.

3.1.7 Predicate "ResDep"

This predicate is used for declaration of activity depending on resource which dispose of some attribute.

$$ResDep(Act, Att, X, Y)$$

This declaration says: There exists activity *Act* and resource attribute *Att* and this activity needs some resource with at least amount *X* of attribute *Att* and at most amount *Y* of attribute *Att* for its execution. Amounts *X* and *Y* are (integer) numbers between 0 and 100 (both included). Those numbers may be interpreted in at least two ways – as a logical value (where 0 stands for false and 100 stands for true) or as a percent.

Both, activity *Act* and resource attribute *Att*, must be declared before.

For composite activities this predicate has following semantics: Resource that is allocated for composite activity *Act* is available for execution of sub-activities of activity *Act* for all the time of execution of activity *Act* and this resource cannot be used "outside" this activity.

3.2 Predicates that make a scheduling problem from a workflow

All the above mentioned predicates are used for describing workflows and resources. To create a scheduling problem we need somehow "instantiate" workflows and resources. For this purpose we need another predicates…

3.2.1 Predicate "Resource"

This predicate is used for declaration of existence of a resource.

$$Resource(R)$$

This declaration says: There exists resource *R*.

3.2.2 Predicate "ResAttrib"

This predicate is used for declaration of resources attributes.

$$ResAttrib(R, A, X)$$

This declaration says: Resource *R* has attribute *A* of amount of *X*.

One resource should have at least one attribute declared.

3.2.3 Predicate "WfInstance"

This predicate is used for declaration of workflow instance in scheduling problem.

$$WfInstance(W)$$

This declaration says: There is an instance W of a workflow.

3.2.4 Predicate "ActInstance"

This predicate is used for declaration of activity instance.

$$ActInstance(I, A, W)$$

This declaration says: There exists instance I of activity A and this activity instance is a part of a workflow instance W.

4 Scheduling

4.1 Checking the scheduling task

4.2 Scheduling

4.3 Output

Schedule – output of scheduling engine – will also be described in declarative format. In fact, this output will be a copy of scheduling engine input with some additional declarations in the end. Those additional declarations will use following predicates.

4.3.1 Predicate "ExecutedAt"

This predicate is used for declaration when an instance of activity is executed.

$$ExecutedAt(I, S, E)$$

This declaration says: Activity instance *I* is executed and starts at the time of *S* and ends at the time of *E*. There may be only one predicate *ExecutedAt* for one activity instance.
Activity instances that don't appear in *ExecutedAt* predicate in the output are not executed at all.

4.3.2 Predicate "ExecutedOn"

This predicate is used for declaration which resources an executed activity instance consumes.

$$ExecutedOn(I, R)$$

This declaration says: During the time of execution of activity instance *I*, this activity instance consumes resource *R*.

In our scheduling problems we may use only unary resources. Therefore there cannot be any non-empty intersection in a timeline for one resource for any activity instances executed consuming this resource.

5 What is missing

Information about:

1) Platform and system requirements
2) Will we use prolog called from C#?
3) How do we do that with time? OK…I have answers for this question. But should I describe it here and should I write the algorithms for translations between our internal (integer) time and real time?
4) How do we do that with resource attributes – how to get them into workflow editor?
5) How will we call the module that will be used for creation a scheduling problem?
6) Do we need to say anything more specific about the scheduling algorithm in advance?
7) What about namespaces (separated for activities, workflows, resources, attributes, … and especially when there are more than one workflow in a scheduling problem and those workflows use the same "name" for activity or something how do we distinguish them from each other)?
8) Should I describe XML format?
9) Should I describe command line execution?
10) Should I describe network communication?
11) I think it's not fair to me to make scheduling engine, resource editor, something else for scheduling problem creation and orchestration module… ☺