SRS - Workflow editor module of the Flow Opt project

This document specifies software requirements on the workflow editor module of the FlowOpt project.

Used terms and remarks

This section describes the terminology used throughout this document, font conventions and other metainformation relating to this document as a whole.

Throughout this document, I sometimes refer to information contained in other parts of the FlowOpt project specification (for example the declarative format for workflows). I therefore assume that the reader is familiar with the rest of the FlowOpt project specification, or at least its parts relevant to this module.

Font conventions
In the text, whenever I mention some term that will be referred to later in the document, it is written in *italics*. Fragments of declarative format code will be written in `courier`. In some cases, I'm not yet certain on the solution of the specific problem – these cases are marked by (TBD) mark and will be resolved later (advice on such problems is especially welcome).

Nested TNA model description
Our workflow model is based on the Nested TNA workflow model introduced in http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.5684&rep=rep1&type=pdf. Any information on this model not provided by this document can be found there.

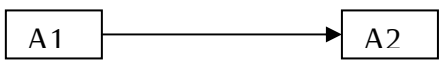
Declarative format
By declarative format, I mean the data format used to pass a workflow from this module to the scheduling engine module. Its complete description is in the project specification. In this document, I use n-ary `Activity` predicate instead of unary simply to save space, therefore instead of:

```
Activity(A1, Duration of A1, Cost of A1, …)
Activity(A2, Duration of A2, Cost of A2, …)
Activity(A3, Duration of A3, Cost of A3, …)
```

I only write

```
Activity(A1,A2,A3)
```

Also, for every arc there will be a predicate with its precedence constraint like in following case:



Produces a predicate like
`Temporal(A1,A2,5,10,ES)`

I don't explicitly write these precedence predicates, again to save space. All temporal predicates that are explicitly written are independent of individual arcs (they are a part of specific pattern transformation).

Various remarks
For the sake of this document, clicking refers to left clicking; dragging refers to dragging the mouse while holding the left mouse button.

Application overview

This application will be a part of the Flow Opt project – specifically it will be a graphical workflow editor module, which will allow the user to create or import a workflow, edit it, save it and use it for scheduling by other Flow Opt modules. Emphasis will be on user friendliness, usability and completeness with regard to this specification.

Hardware and software requirements

The application will be written in C# language and therefore, the user will be required to have .NET framework 3.5 installed to run this application. As for hardware requirements, a common up to date PC (2GHZ CPU, 1GB RAM) should be sufficient to run the application smoothly. Since the FlowOpt project's modules will communicate via network, the user's computer will also have to have network connectivity in order to use other FlowOpt modules remotely. Since network traffic should be minimal, any current connection (256 Kbit/s and faster) should be sufficient.
Obviously in order to access functions provided by other FlowOpt modules locally, the user will have to have those installed as well and therefore meet the HW and SW requirements for these modules. Those can be found in FlowOpt specification.

Workflow model

Our workflow model is based on the Nested TNA model, but there are some differences. The user will be able to perform all operations relevant to the Nested TNA model (parallel / alternative / serial decomposition), plus he will be able to arbitrarily specify precedence, logical and synchronization constraints. This means that the user can choose any two vertices in the workflow graph and specify a precedence, logical or synchronization constraint for them. All this will be thoroughly specified later in this document.

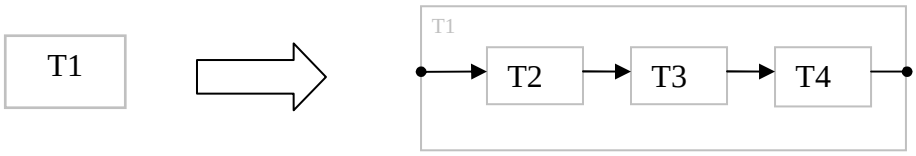
Domain model

The application will serve as graphical workflow editor; hence the main object of interest will be a workflow. Workflow in this case can be considered an directed graph G(V, E). Vertices (V) will be called *tasks*. Arcs (E) will be of three kinds: *precedence*, *logical* and *synchronization* arcs. All of these will be further described below.

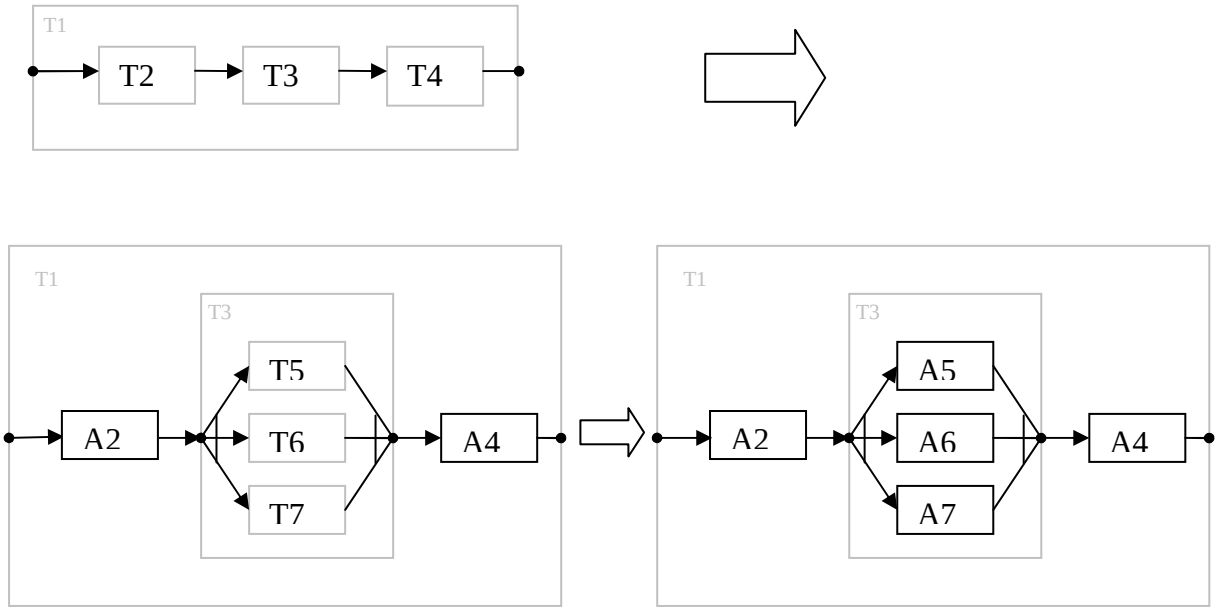
Tasks and Activities

Activities and tasks form the core of the workflow. Activities effectively say what should be done – they are the elementary components which build the workflow and they cannot be further decomposed. Tasks on the other hand serve as placeholders for activities or other tasks and store the structure of the workflow. Main difference between an activity and a task is the fact that activities are scheduled for execution when using the scheduler module of Flow Opt project, whereas tasks are not – they only inform the scheduler of the workflow structure. Also, an activity has a number of properties that the user can fill – for example cost, duration etc. Tasks have no such properties (only an identifier).

One task can be thought of as a slot that can contain either a single activity, or a *nest*. A nest is created by decomposing a task (which is then called a *parent task*) using parallel, alternative or serial decomposition, which will be thoroughly specified later in the document. For sake of this section, a nest is a set of tasks which are to be executed in place of the parent task.
If a task doesn't contain any activity or task, it is an *empty* task. Note that an empty task means an incomplete workflow – it is not the same as say an activity that doesn't do anything, the user will have to assign an activity to all empty tasks before the workflow is considered complete (ready for scheduling).
When creating a new workflow, the user will start with a single task (called *initial task*), representing the whole workflow process. He will then decompose this initial task into more tasks, forming the structure of the workflow. To complete the workflow, the user will assign activities to empty slots.
The process is illustrated by the following picture (empty tasks are gray rectangles, tasks with activities are black rectangles):



Here the user started with one initial task T1 – this task is now empty. Then the user decomposed T1 into three tasks T2, T3 and T4. These three tasks now form T1's nest – T1 is no longer an empty task, however T2, T3 and T4 are empty tasks. The user may now either decompose them to fill them with their own nests, or he can assign them an activity to fill them.
In the following figure, T2 and T4 are filled with an activity (A2 and A4 respectively), while T3 is decomposed into three more tasks T5, T6 and T7, which are subsequently filled with activities (A5, A6 and A7) to complete the workflow.



The resulting workflow has 7 tasks and 5 activities. Initial task T1 is filled with its nest composed of T2 (filled with activity A2), T4 (filled with activity A4) and T3 filled with its

own nest composed of tasks T5, T6 and T7 (filled with activities A5, A6 and A7 respectively).
The nature of decomposition operation is not relevant at this point, only the difference and use of activities and tasks is relevant.

Precedence, Logical and Synchronization arcs

Arcs in the workflow graph represent various constraints on tasks. Some arcs are created automatically through decomposition (these are only precedence arcs), but the user can arbitrarily create arcs of all kinds.
All arcs can be only connected to tasks (remember, activities are not considered vertices in the workflow graph). Note that tasks have their start and end times specified implicitly – start time of a task is the start time of its first (time wise) scheduled activity, while end time of a task is the time of its last (time wise) scheduled activity. Therefore, if a task only has one activity within itself, the start and end times of this task and its activity are identical and the arc effectively connects to this activity.
Of the three kinds of arcs, precedence arcs are probably the most common – they simply represent order within the workflow: when task T1 is connected to task T2 using a precedence arc, it means that T1 has to be completed before T2 can start.
Logical arcs serve to establish some logical constraints on tasks. There are three kinds of logical arcs: equivalence, implication and mutex. Suppose we have tasks T1 and T2. Then a logical arc (T1, T2) with equivalence constraint means that T1 is to be scheduled for execution if and only if T2 is scheduled for execution (time doesn't matter, only the fact that they are either both scheduled, or none of them is). A logical arc (T1, T2) with an implication constraint means that if T1 is scheduled for execution, then T2 also has to be scheduled for execution. Finally, a logical arc (T1, T2) with a mutex means that if T1 is scheduled for execution, T2 cannot be scheduled for execution and vice versa – the two tasks are to be mutually exclusive for the scheduler.
Last kind of arcs is the synchronization arcs. These form a synchronization constraint between two tasks and are of two kinds: start-start (SS) and end-end (EE) synchronization. Suppose we have tasks T1 and T2, then a synchronization arc (T1, T2) with SS constraint means that T1 and T2 must start at the same time. A synchronization arc (T1, T2) with EE constraint means that T1 and T2 must end at the same time.

User interface

Below is the basic overview of the main form.

<table>
<tr><td colspan="2">Main menu – File | Edit | … | Help</td></tr>
<tr><td colspan="2">Toolbar – icon shortcuts for aligning, orientation switch, calendar etc.</td></tr>
<tr><td>Drawing area – where the workflow is displayed and edited</td><td>Activity properties – ID, duration, cost, due date etc, When an activity is selected, its data can be seen and edited here</td></tr>
<tr><td colspan="2">Hint box – miscellaneous relevant information for current operation</td></tr>
</table>

Zooming

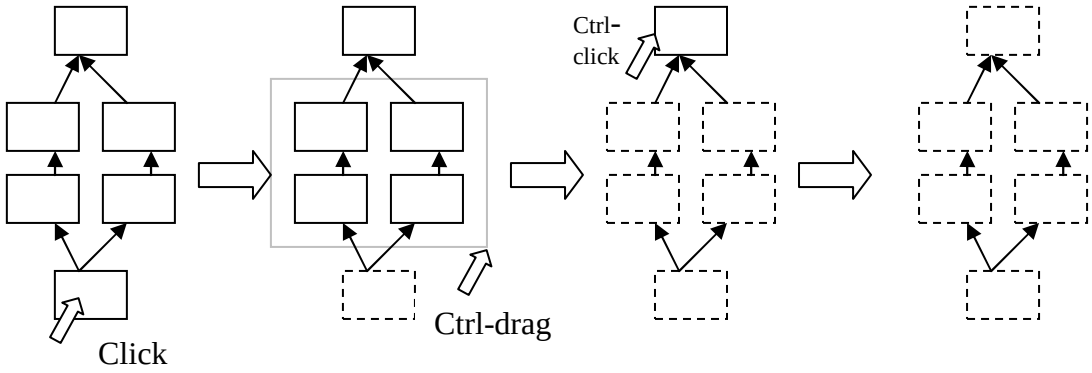
The user will be able to zoom in and out. There won't be a magnifying glass functionality (i.e. the possibility to enlarge just a selected part of the screen, without the rest changing size). Zooming distance will be limited.

Undo

The user will be able to perform an undo operation for all main operations. Its depth will not be unlimited. In some cases where undo operation is not well defined, it may not be available (TBD).

Selection

The user will be able to select a single activity by left clicking on it (or more precisely on its task). By doing so, its properties will be displayed in the activity property form, where they can be viewed and edited.
By ctrl-clicking on activities, they are added/removed to/from current selection. By dragging the mouse when nothing is selected, a rectangle appears indicating selection – upon releasing the mouse button, all activities within the rectangle will be selected. By ctrl-dragging the mouse, multiple activities can be added to selection. When multiple activities are selected, their properties cannot be edited; the activity property form is only active when a single activity is selected. Selected activities are highlighted (dashed line). Same applies for arc selection.



Activity and arc properties

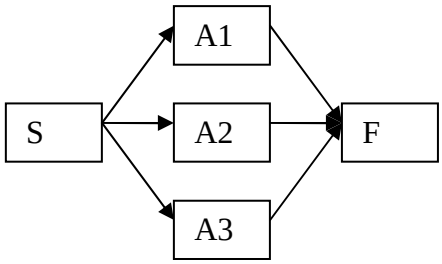
Both activities and arcs will have certain properties, which the user will be able to view and edit. In case of activities, these properties include for example ID, duration, cost, due date, late cost, resource requirements etc. There will be a special panel for viewing and editing these properties (see main form overview). Upon selecting a single activity, its properties are loaded into this form where the user may view and edit them.
As for arcs, they have various properties based on the type of an arc – precedence arcs have no properties, logical and synchronization arcs have a single property: type of their operation. The user may input this information by double clicking on the arc, which will cause an input field to appear, allowing user to enter the property.

Resources
One of activity properties to take special note of are its resource requirements. These will be specified in the form of necessary attributes the resource must have in order for it to be able to perform the activity. Specifically, the resource editor (another FlowOpt module) will allow the user to define a set of resource attributes. In this (workflow editor) module, the user will be able to assign requirements on those attributes to any activity.
These requirements will be entered as a percentage (1 to 100%). For example, in resource editor the user might define resource attributes strength, sharpness, durability and precision. For an activity “Saw lumber” the user may specify a requirement of 75% sharpness, 50% durability and 25% precision. For an activity “Carry planks to warehouse” the user may specify 100% strength attribute requirement. Finally, for activity like “Make a chair” the user may enter 30% durability and 80% precision requirement.
There will be special dialogue in the activity property form to allow the user to enter the above described data.

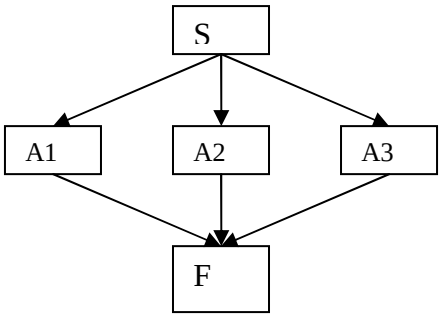
Choice of orientation

In any moment, the user will be able to switch between two orientations – horizontal (left to right) and vertical (top to bottom). The following figure illustrates both cases:

Horizontal orientation:



Vertical orientation:



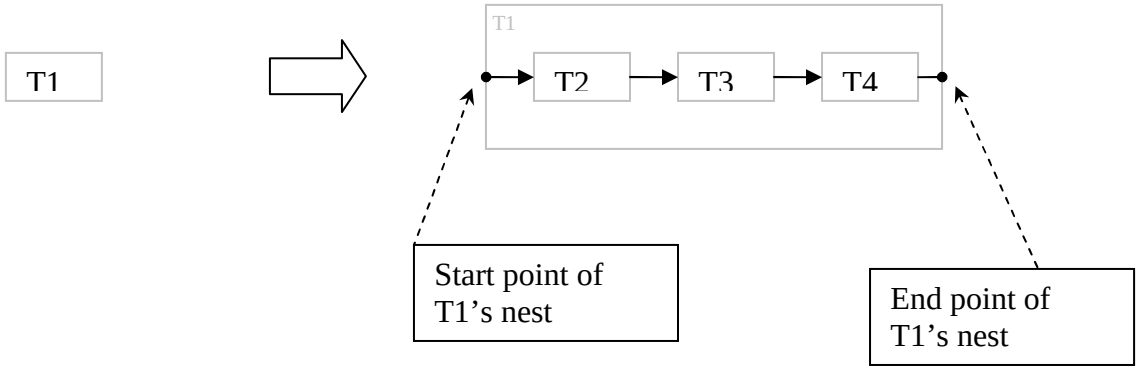
Editing

In the beginning, the user starts with a workflow consisting of a single task (vertex)[1]. For sake of this specification, this task will be called the *initial task* and workflow consisting of this single task will be referred to as *initial workflow*. On this elementary workflow, the user can apply several editing actions.
To change the workflow structure, the user must perform decomposition – one task is decomposed into several new tasks, which are to be executed in a certain way in place of the original task. Upon decomposition, a task becomes a *parent* task. All task s created by the decomposition belong to the parent task's *nest*. Tasks that are not parent, i.e. they have not been decomposed, will be referred to as *empty* tasks.
Every nest has two special points to take note of – the *starting point* and the *end point*. The former represents the point at which this nest starts; the latter represents the point at which the nest ends (time wise). These points are marked by a black dot on edges of the nest (see figures below). These points serve to connect tasks within the nest to the parent composite task – the start point corresponds to start of the parent task and the end point corresponds to end of the parent task.
Besides decomposition, the user can also arbitrarily specify binary temporal, logical and synchronization constraints, as described earlier in this document.

All editing options are further described here:

- **Serial decomposition** – the user can split a single task into multiple tasks, which will be executed one after another in place of the original task.
  Here T1 is decomposed into tasks T2, T3 and T4. T1 thus becomes a parent task, with T2, T3 and T4 in its nest. The editor will visualize the nest like in the figure – as a grey rectangle over all the nested tasks, with a name of the parent task. It will be possible to turn this visualization off.

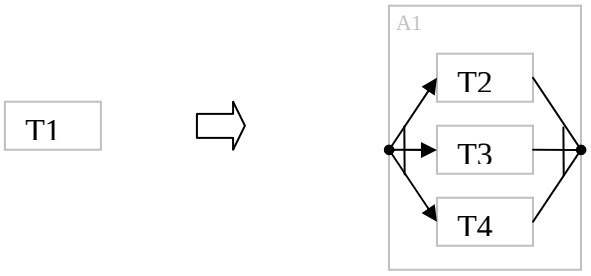
[1] In the formal Nested TNA model, the user starts with two tasks connected by an arc, however unlike the formal model, we decompose the vertices rather than arcs, hence the slight change of model.



Declarative format transcription:

```
Activity(A1)
=>
Activity(A1,A2,A3,A4)
Temporal(A1,A2,0,inf,SS)            //start of A1's nest to start of A2
Temporal(A2,A3,0,inf,ES)            //temporal arcs between created
Temporal(A3,A4,0,inf,ES)               activities
Temporal(A4,A1,0,inf,EE)            //end of A4 to end of A1's nest
Decomposition(A1,[A2,A3,A4],AND)    //nest creation
```

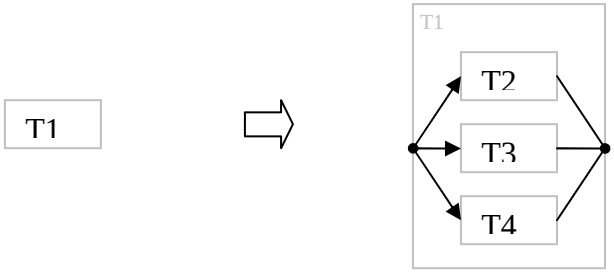
- **Parallel decomposition** – the user can split a single task (vertex) into multiple tasks, which will be executed in parallel in place of the original task (produces AND split and join). In this example, T1 is decomposed into T2, T3 and T4. Arcs leading to and from these tasks are marked with a line crossing them – this indicates the parallel (AND) relation.



Declarative format transcription:

```
Activity(A1)
=>
Activity(A1,A2,A3,A4)
Decomposition(A1,[A2,A3,A4],AND)
Temporal(A1,A2,0,inf,SS)      Temporal(A2,A1,0,inf,EE)
Temporal(A1,A3,0,inf,SS)      Temporal(A3,A1,0,inf,EE)
Temporal(A1,A4,0,inf,SS)      Temporal(A4,A1,0,inf,EE)
```

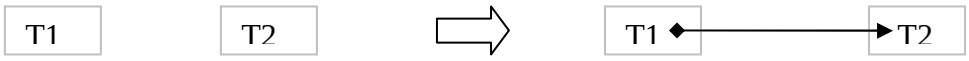
- **Alternative decomposition** – the user can split a single task into multiple tasks, out of which a single task will be executed in place of the original task (produces XOR split and join). In the following figure, T1 is decomposed into T2, T3 and T4. Notice that unlike in the previous case, there is no line crossing arcs incident with T2, T3 and T4 – this means alternative decomposition.



Declarative format transcription:

```
Activity(A1)
=>
Activity(A1,A2,A3,A4)
Decomposition(A1,[A2,A3,A4],XOR)
Temporal(A1,A2,0,inf,SS)      Temporal(A2,A1,0,inf,EE)
Temporal(A1,A3,0,inf,SS)      Temporal(A3,A1,0,inf,EE)
Temporal(A1,A4,0,inf,SS)      Temporal(A4,A1,0,inf,EE)
```

- **Adding a precedence arc between two tasks** – the user can create a new precedence arc between two tasks. In this example we specify a precedence arc for tasks T1 and T2.
  Note that the arc's starting point is different from the precedence arcs which are created through decomposition and it is also marked (diamond) to indicate that this temporal arc was created arbitrarily, not through decomposition.

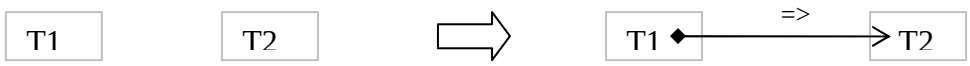


Declarative format transcription:

```
Activity(A1,A2)
=>
Activity(A1,A2)
Temporal(A1, A2, a, b, ES)
```

- **Adding a logical arc between two tasks** – the user can also specify a logical arc for a couple of tasks T1 and T2.
  These constraints will be visualized similarly to the precedence constraints – as arcs. These arcs will not be directed and may be drawn with different color. To prevent confusion, the user will be able to only show one kind of arcs (temporal / logical / synchronization), show one and draw the others in the background grayed out, or show both kinds of arcs normally.

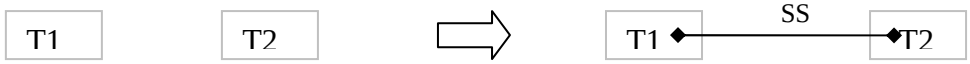
In this example, we specify an implication constraint for tasks T1 and T2. As in the case of arbitrary precedence arcs, arbitrary logical arcs also start in a different point and their start is also marked (diamond) in order to distinguish them from arcs created by decomposition.



Declarative format transcription:

```
Activity(A1,A2)
=>
Activity(A1,A2)
Logical(A1, A2, =>)
```

- **Adding a synchronization arc between two tasks** - The user can add a synchronization arc between two tasks. These are again marked differently from arcs created through decomposition, as illustrated by following figure. Here we specify a start-start synchronization constraint for tasks T1 and T2.

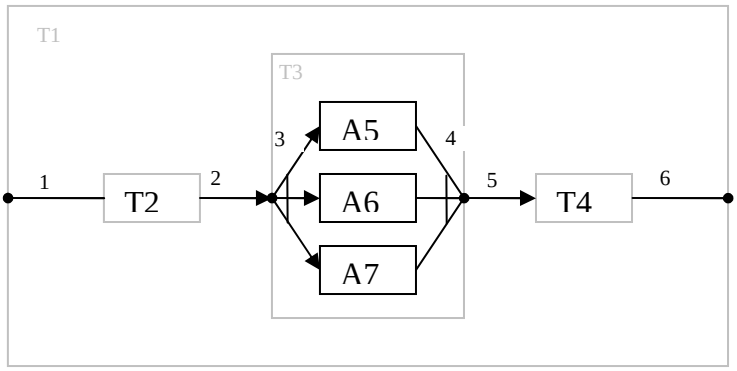


Declarative format transcription:

```
Activity(A1,A2)
=>
Activity(A1,A2)
Logical(A1, A2, =>)
```

Transcribing arcs into declarative format

There are many cases where arc constraint transcription into declarative format may not be obvious. In case an arc directly connects two activities, transcription simply produces a single predicate (either `Temporal` or `Logical` depending on arc type).
However, some arcs also connect activities with starting or end points of a parent activity, as in following figure (arcs that will be referred to below have been numbered):



There are basically four special cases that may happen (I assume all temporal constrains in the picture are [0, inf] since exact values don't matter here):

- Arcs leading from the starting point of a parent activity (arcs 1, 3 in the figure) – those produce a temporal constraint of the SS type (start – start) like so:
  `Temporal(A1,A2,0,inf,SS)`
  `Temporal(A3,A5,0,inf,SS)`
- Arcs leading to an ending point of a parent activity (arcs 4, 6 in the figure) – those produce a temporal constraint of the EE type (end – end) like so:
  `Temporal(A5,A3,0,inf,EE)`
  `Temporal(A4,A1,0,inf,EE)`
- Arcs leading to starting point of a composite activity (2 in the figure) – those simply produce a temporal constraint of the ES type (end – start). Notice that these arcs were already described, since they actually do connect two activities, for example arc 2 connects A2 and A3 and produces following constraint:
  `Temporal(A2,A3,0,inf,ES)`
- Arcs leading from an ending point of a composite activity (5 in the figure) – same thing as in previous case, these arcs again just connect two activities. For example arc 5 connects activities A3 and A4, and therefore simple ES type temporal constraint is sufficient, like so:
  `Temporal(A3,A4,0,inf,ES)`

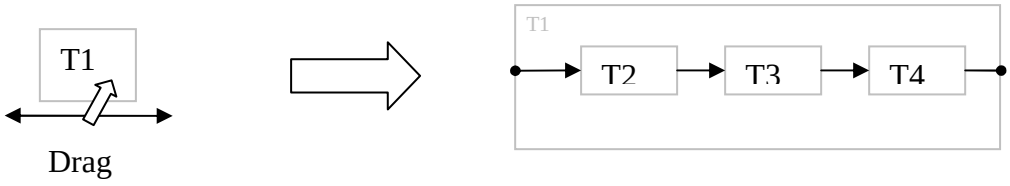
Description of editing process

Here is a step-by-step example of creation of a simple workflow to illustrate the process. It assumes the horizontal orientation (for vertical orientation, the left-right dragging is replaced by up-down dragging and vice versa).

1) The user starts with a single task (the initial task), which represents the entire workflow.

Declarative format transcription:
`Activity(A1)`

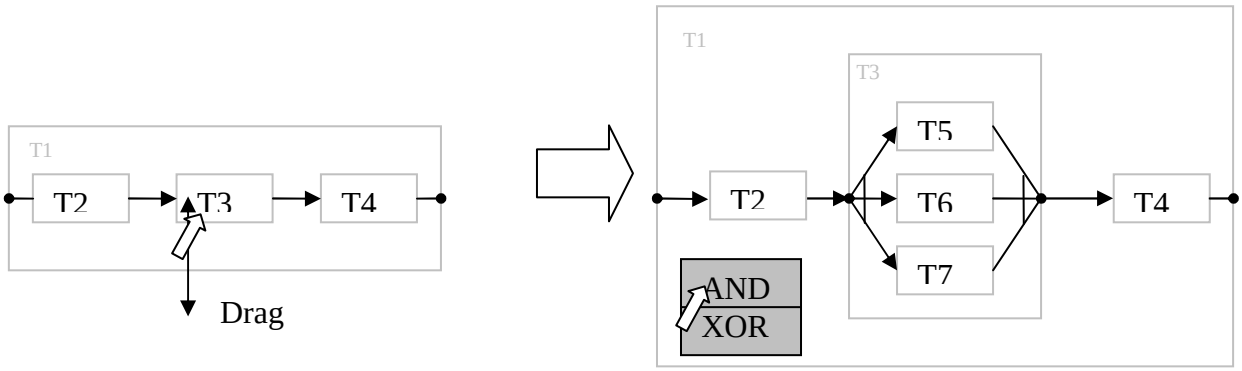
2) Serial decomposition application – the user clicks on the task, which he wants to decompose (in this case T1) and drags the mouse while holding the left mouse button. Dragging to the right adds more tasks to the decomposition, dragging to the left removes tasks from the decomposition (in this case, the user splits the T1 task into three serially executed tasks T2,T3 and T4):



Declarative format transcription:

Activity(A1,A2,A3,A4)[2]
Decomposition(A1,[A2,A3,A4],AND)
Temporal(A2,A3,0,inf,ES)
Temporal(A3,A4,0,inf,ES)

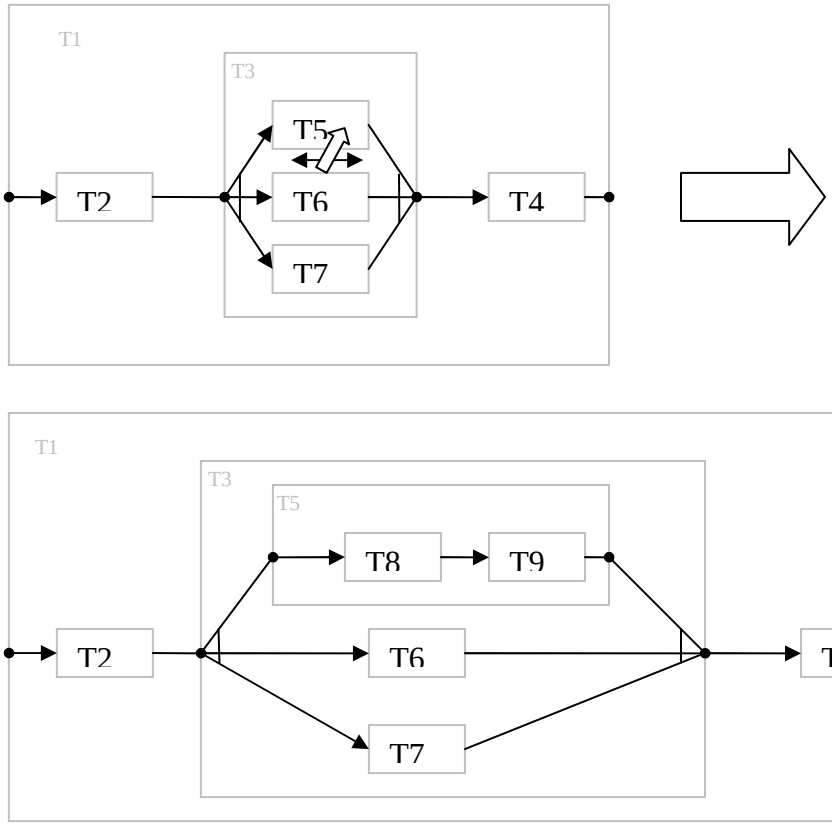
3) Parallel decomposition application – The user clicks on the task, which should be decomposed (in this case T3) and by holding the left mouse button and dragging the mouse he specifies the number of tasks in the decomposition similarly as in the serial decomposition case – dragging up adds more tasks into the decomposition, dragging down removes tasks from the decomposition.
Here the user decomposed the T3 task into three tasks T5, T6 and T7, which should be executed in parallel in place of the original T3 task. When the user releases the mouse button, a pop-up menu automatically appears with the choice of either AND split (parallel decomposition) or a XOR split (alternative decomposition) – when the user wishes to perform an alternative decomposition, the process is the same as with parallel, except for selecting XOR split in the pop-up menu.



Declarative format transcription:
Activity(A1,A2,A3,A4,A5,A6,A7)
Decomposition(A1,[A2,A3,A4],AND)
Temporal(A2,A3,0,inf,ES)
Temporal(A3,A4,0,inf,ES)
Decomposition(A3,[A5,A6,A7],AND)

4) Further serial decomposition. This time the user chooses to decompose T5 into two tasks T8 and T9:

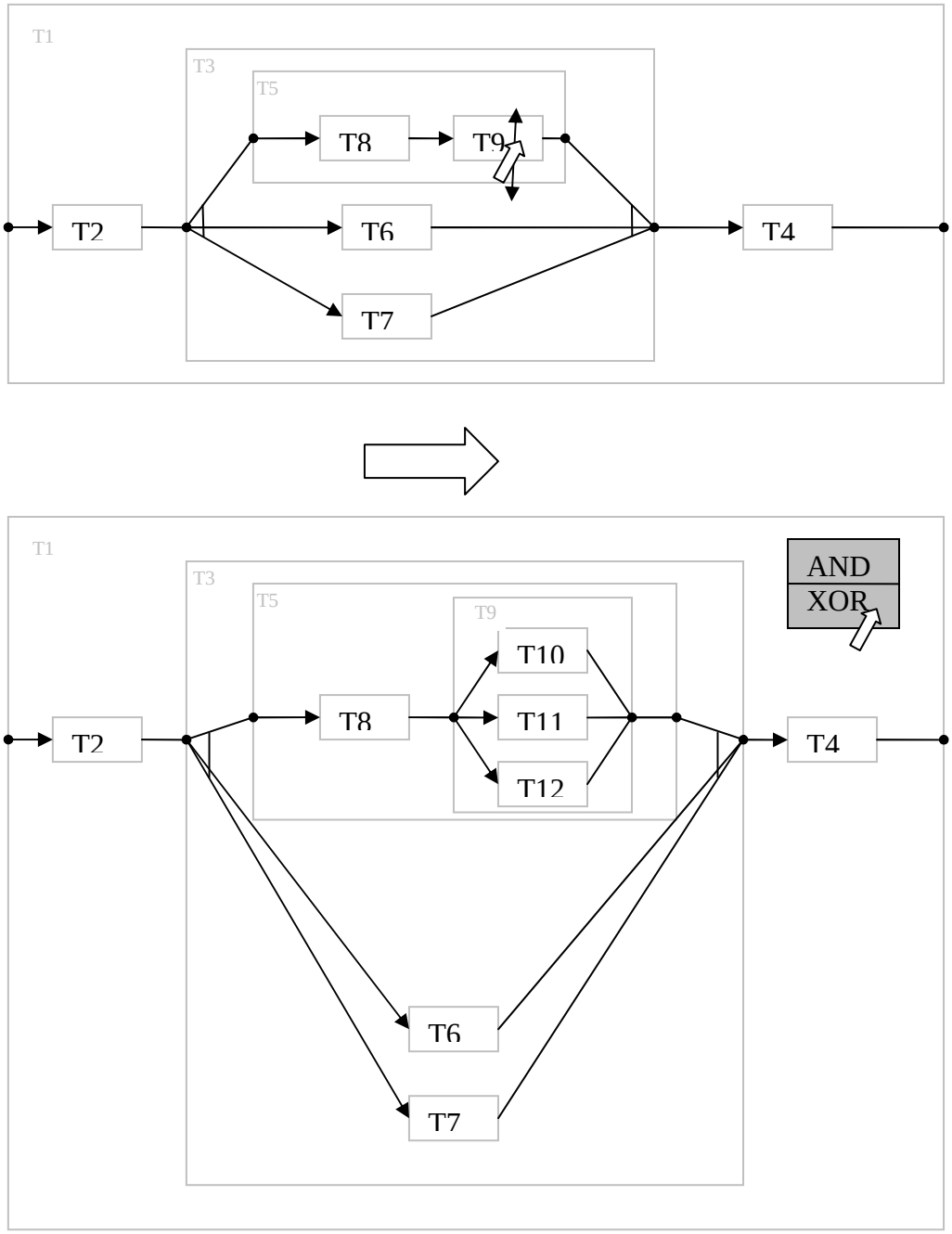
[2] To save space, I use n-ary predicate instead of unary for specifying activities.



Declarative format transcription:

```
Activity(A1,A2,A3,A4,A5,A6,A7,A8,A9)
Decomposition(A1,[A2,A3,A4],AND)
Temporal(A2,A3,0,inf,ES)
Temporal(A3,A4,0,inf,ES)
Decomposition(A3,[A5,A6,A7],AND)
Decomposition(A5,[A8,A9],AND)
Temporal(A8,A9,0,inf,ES)
```

5) Alternative decomposition – As stated earlier, the process is similar to that of parallel decomposition, except in the pop-up menu XOR is selected indicating that the user wishes for an alternative decomposition, not parallel. Here the user chose to decompose the T9 task into three tasks T10, T11 and T12. When the T9 task should be performed in the original workflow, exactly one of the tasks T10, T11, T12 will be executed in the new workflow:



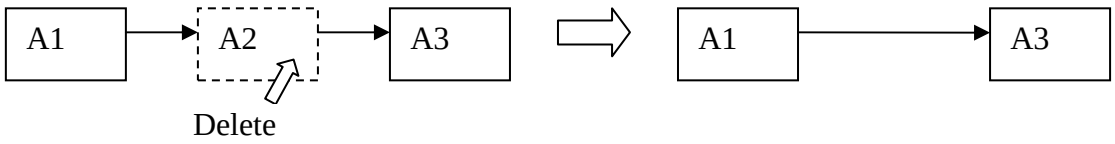
Declarative format transcription:

```
Activity(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12)
Decomposition(A1,[A2,A3,A4],AND)
Temporal(A2,A3,0,inf,ES)
Temporal(A3,A4,0,inf,ES)
Decomposition(A3,[A5,A6,A7],AND)
Decomposition(A6,[A8,A9],AND)
Temporal(A8,A9,0,inf,ES)
Decomposition(A9,[A10,A11,A12],XOR)
```

After this, the user would still have to fill the empty tasks (in this case T2, T4, T6, T7, T8, T10, T11, T12) with activities to be able to schedule this workflow. This example was to illustrate the decomposition process only.

Another way to do editing actions
An alternative way to perform decomposition will be through right clicking on a task the user wishes to decompose and selecting “Decompose this task” option from the pop-up menu that appears. This will produce a dialog allowing the user to enter the number of tasks that should be in the nest and type of decomposition. This option is mainly intended for large-scale decomposition, where dragging would not be convenient for the user.
For the same reason, there will also be a special dialog for creating temporal, logical and synchronization constraints. This dialog will allow the user the enter IDs of two tasks that should be connected by an arc (and of course type of the arc) to make connecting arcs that are far away from each other more convenient.

Removal of activities and tasks
The user can remove the selected activities within one task by pressing the delete key or by selecting the right option from right-click pop-up menu. Once an activity is deleted, the task which contained it becomes empty again – it can be filled by another activity or decomposed further.
Tasks can also be deleted – this deletes everything within them (all tasks, activities) – a dialog will appear requesting user confirmation. For every deleted task, all incident arcs are automatically deleted too.
In case the user removes a task within a sequence, it is removed together with (two) incident arcs. If the removed activity had two neighbors, a new precedence arc is created to connect them as in the following figure.



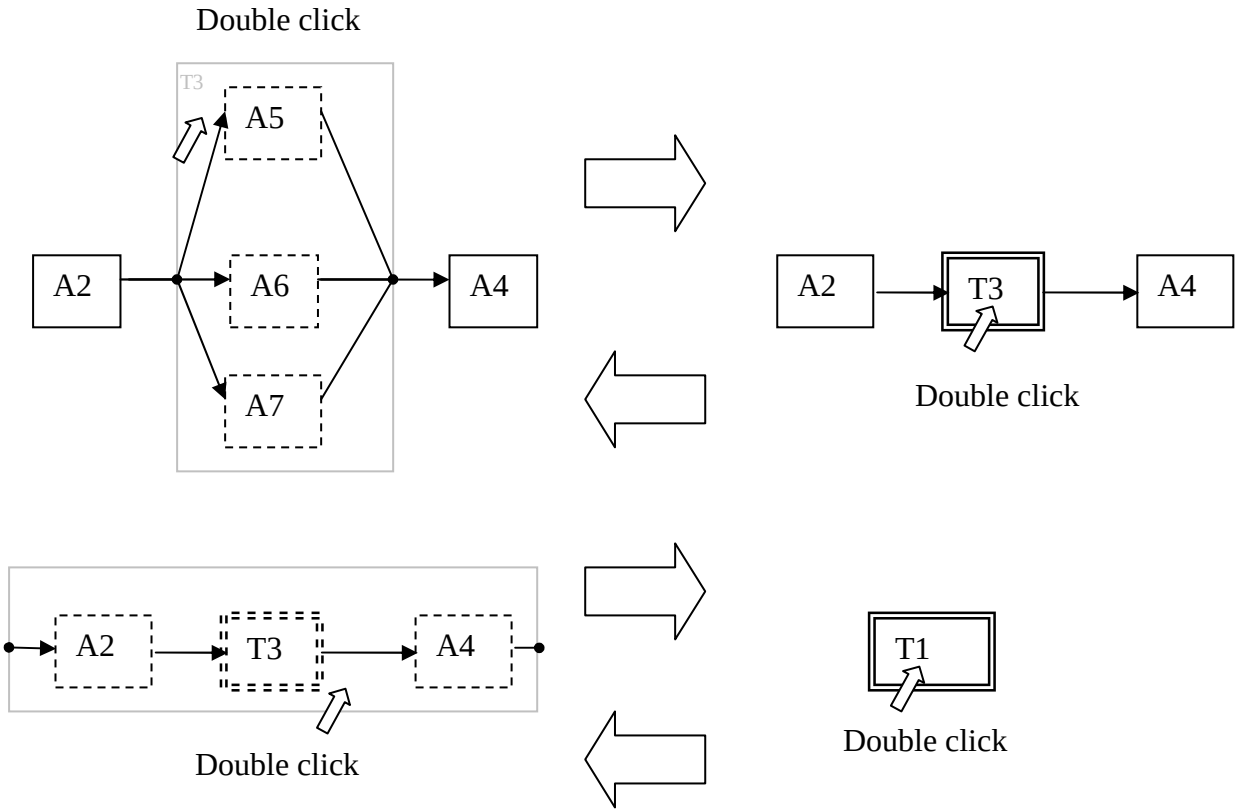
Hierarchy

The user will be able to work with the natural hierarchy induced by the Nested TNA model. Since every decomposition creates a nest within decomposed task, we can see a tree-like hierarchy with the initial task in its root and elementary tasks in its leaves. The user will be able to show or hide any subset of the nests he created, as well as expand a single nest into new window as a separate workflow, edit it and save it either into the original workflow in place of the expanded nest, or as a completely new workflow.

Process description
When the user moves mouse pointer over any task, all other tasks belonging into the same nest are highlighted automatically. Upon double clicking on the parent task, the nest is collapsed into the parent task, marked in a special way indicating that it is not an empty task, but a composite one containing a collapsed nest of tasks. Upon double clicking on a collapsed composite task, it is once again replaced by the nest of tasks within it. This process can be applied arbitrary number of times, giving two extremes: fully expanded workflow, where

every activity and every task is visible and fully collapsed workflow, where only the initial task is displayed.
The user can also right click anywhere in a nest (or on a composite task) and select “Expand in a new window” option to open a separate window containing just the selected nest.

Example
In this example case, the user created a network consisting of 5 activities A2, A4, A5, A6 and A7. Activities A5, A6 and A7 form a nest decomposed from task T3, so upon double clicking anywhere in T3’s nest, it is collapsed as seen in the first figure (arrow to the right). On the other hand, when T3 is collapsed, it can be expanded again by double clicking on it.
The second figure illustrates the same thing one hierarchy level higher – since A2, T3 and A4 are also a nest created from T1 (the initial task), the user can collapse this nest too. Upon double clicking within T1’s nest, it is collapsed and only the initial (now also composite) task T1 is displayed. Double clicking on it will now expand this task’s nest.



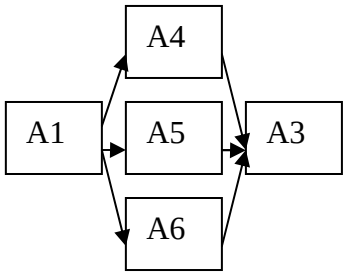
Manipulating tasks

It will also be possible to do the following:

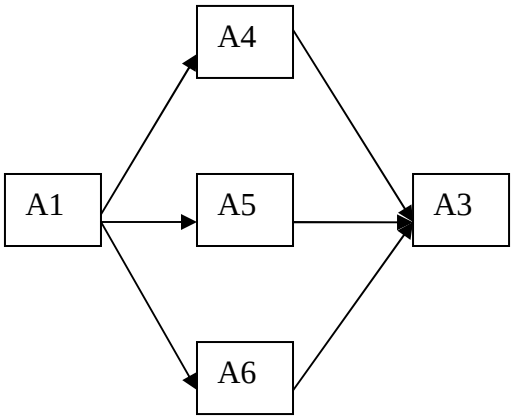
- Add more tasks to a nest – right click + “Add more tasks to this nest”. A dialogue appears where the user may enter a number of tasks to be added.
- Change the order of tasks in a nest (“swap” tasks) – changes the execution order in case of sequence, for parallel / alternative decomposition it is only for user’s convenience. It will probably be done by using a special “cursor mode” for swapping tasks within the same nest.

Padding
The user will be able to set the horizontal and vertical padding between tasks, i.e. how much space is there between tasks on the screen.

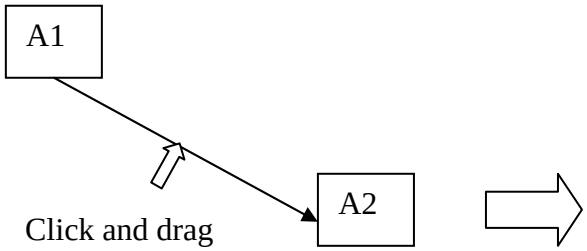
Example:
Low padding

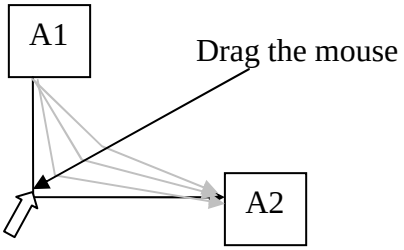


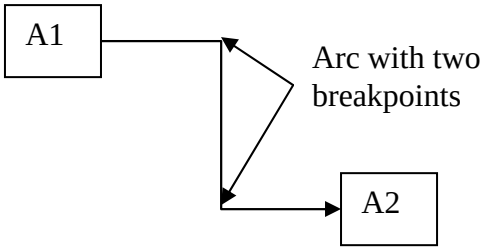
Higher padding



Transforming arcs
The user will be able to transform arcs which were not added through decomposition by adding active breakpoints on them like so:







Calendar

The application will also provide a simple calendar as a means to specify when an activity can be scheduled (for ex. the user will be able to specify that activity A1 can only be scheduled on Mondays from 10:00 to 12:00).

Import/Export of the workflow

The user will be able to import/export a workflow from/to the YAWL format (http://www.yawlfoundation.org/). Since the YAWL format is significantly more complex than our workflow format, there will be some modifications on the workflow begin imported from YAWL – the user will be notified of these modifications and, in case it is needed, the users feedback will be requested on how should the YAWL workflow be transformed to fit into our workflow format. The main goal here is to create a workflow visually resembling the one being imported from YAWL, not to create an identical workflow in our format (that is impossible due to said differences between our format and YAWL).

Table of Contents