

Logistics Specification and Analysis Tool (Eclipse LSAT™) User Guide

Version v0.6

Table of Contents

1. Eclipse LSAT™ IDE	2
1.1. Logistic perspective.....	2
1.2. Motion Calculator	3
2. Logistics Specification	4
2.1. Create New Project	4
2.2. Create Machine Specification	5
2.2.1. Extended/Advanced Machine Specification	7
2.3. Create Settings Specification	9
2.3.1. Plotting a timing distribution.....	11
2.3.2. Extended/Advanced Settings Specification.....	11
2.4. Create Activity Specification	14
2.4.1. Extended/Advanced Activity Specification.....	16
2.4.2. Constraints for ALAP and ASAP scheduling actions.....	19
2.4.3. Plotting a move	24
2.5. Create Activity Dispatching Specification.....	25
2.5.1. Extended/Advanced Activity Dispatching Specification	26
2.5.2. Specifying the number of iterations.....	27
2.6. Create Supervisory Controller Specification	27
2.7. Preferences	28
2.7.1. Editors Configuration.....	28
3. Parameterized Activities Details	30
3.1. Defining Parameterized Activities	30
3.2. Parameter Types	30
3.3. Using Parameterized Activities	30
3.4. Parameter Assignment Styles	30
3.5. Best Practices.....	31
3.6. Examples.....	31
3.7. Benefits	31
3.8. Common Issues	31
3.9. Validation Rules	31
3.10. Quick Syntax Examples.....	32
4. Graphical Viewers	33
4.1. Add Viewpoints Selection	33
4.2. Generate Graphs	33
4.2.1. Peripheral types in machine	33
4.2.2. Resources and peripherals in machine.....	34
4.2.3. Axis position graph for peripheral	35
4.2.4. Symbolic position graph for peripheral	37

4.2.5. Motion profile settings table for peripheral	45
4.2.6. Physical location settings table for peripheral	45
4.2.7. Activity diagram	46
4.3. Location of Generated Graphs	49
4.4. Navigation among graphs	50
4.4.1. Navigate from activity diagram	51
4.4.2. Navigate from axis position graph	51
4.4.3. Navigate from symbolic position graph	52
5. Schedule Activities	53
5.1. Scheduling & Configurations	53
5.1.1. Product Trace Analysis	56
5.2. Gantt Charts	56
5.2.1. Gantt chart	57
5.2.2. Gantt chart with critical path	60
5.2.3. Gantt chart using stochastic impact analysis	61
5.2.4. Filtering	61
6. Throughput per Resource Editor	63
6.1. Opening the Throughput per Resource Editor	63
6.2. The Resource Actions Table	63
6.3. The Outline View	65
6.4. Calculating the Throughput per Resource	65
7. Generating an Optimal Activity Dispatching Sequence	68
7.1. Generating a Dispatching File for Maximum Throughput	68
7.2. Generating a Dispatching File for Minimum Makespan	72
7.3. Viewing Makespan/Throughput Values	75
8. Conformance Check	76
8.1. Add Trace Point	76
8.2. Conduct Conformance Check	77
8.3. Filter Trace Point	81
9. Generate Max-Plus Specification	82
10. Generate GraphML from CIF specification	85
11. Application Programming Interface (API)	87
11.1. REST calls	87
11.2. JSON format	88
11.3. Python API client	89
11.4. Java API client	90
11.5. C++ API client	90
12. Twilight Manufacturing System: a showcase	92
13. Python Execution Engine	95
13.1. Introduction	95
13.2. Usage	96

13.3. Installation	97
13.3.1. Install from the P2 repository (Eclipse GUI)	97
13.3.2. Notes and verification	97
14. Release Notes	99
14.1. Eclipse LSAT™ v0.6 (2026-07-31)	99
14.1.1. Enhancements	99
14.1.2. New Documentation	100
14.1.3. Bug Fixes	100
14.1.4. Architecture/Code Improvements	101
14.2. Eclipse LSAT™ v0.5 (2026-02-06)	101
14.3. Enhancements	101
14.4. Bug Fixes	102
14.5. Eclipse LSAT™ v0.4 (2025-08-08)	103
14.6. Eclipse LSAT™ v0.3 (2024-02-15)	103
14.7. Eclipse LSAT™ v0.2 (2023-02-07)	103
14.8. Eclipse LSAT™ v0.1 (2022-11-01)	104

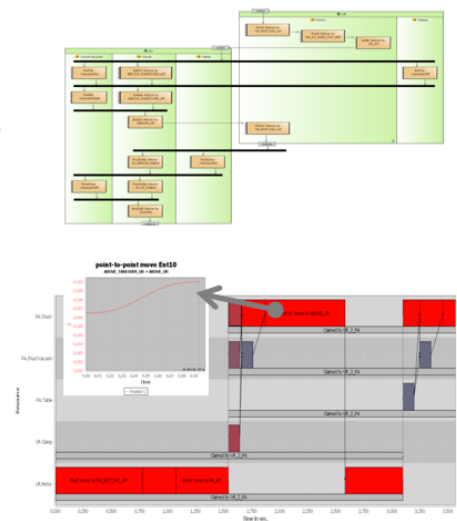
Eclipse LSAT™, Logistics Specification and Analysis Tool, is a tool for rapid design-space exploration of product logistics in flexible manufacturing systems. Eclipse LSAT™ has been developed by TNO-ESI in collaboration with ASML, VDL ETG, Nexperia ITEC and TU/e.

The tool enables lightweight modeling of system resources, system behavior, and timing characteristics. The tool provides various visualizations to explore the system behavior. Eclipse LSAT™ provides efficient performance analysis to compute the maximum system performance and to identify bottlenecks by exploiting the structure of the models.

The figure below shows the main functionality of Eclipse LSAT™.

Main functionality

- **Specification (textual and graphical)**
 - Machine layer: resources, peripherals, locations, paths, profiles,
 - Activity layer: activities (transfers)
 - Logistics layer: state machines, activity dispatch sequence
- **Analysis**
 - Gantt charts
 - Critical path
 - Throughput per resource
- **Optimization**
 - Throughput / makespan-optimal 'good weather' sequences

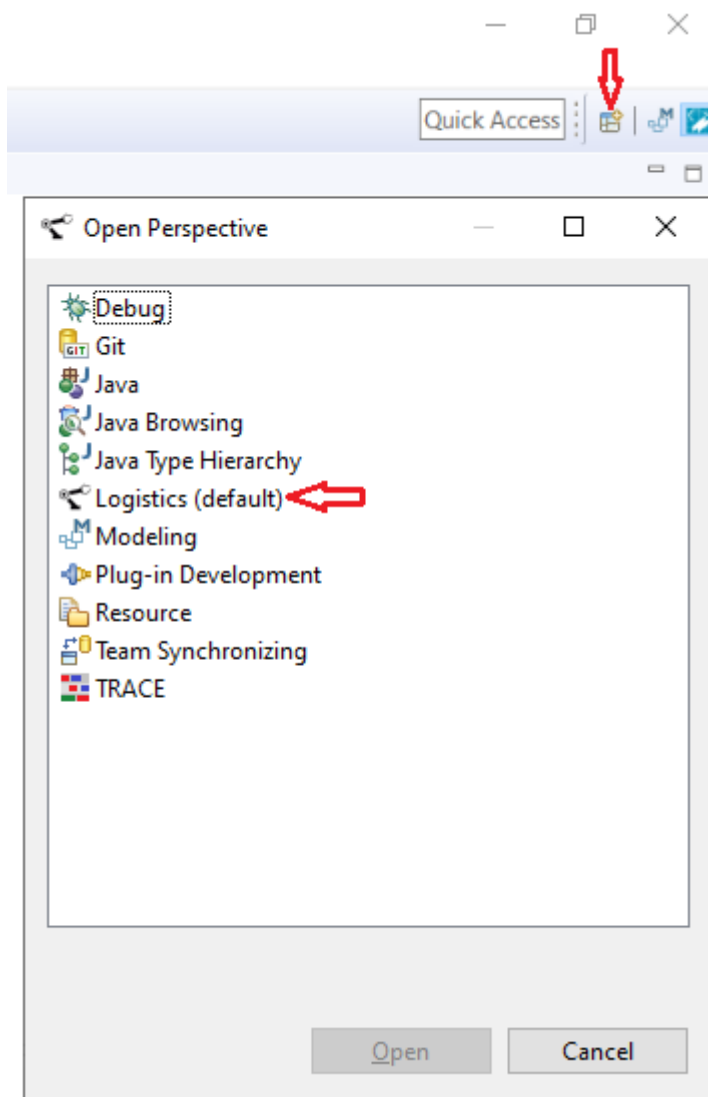


Chapter 1. Eclipse LSAT™ IDE

1.1. Logistic perspective.

Once you have installed Eclipse LSAT™, you can choose the *Logistics* perspective following the instructions as below:

- In Eclipse: select **Window** › **Perspective** › **Open Perspective** › **Other...** › **Logistics**
- Or, click to the icon of perspective (right-up of the window) and select *Logistics* as it is shown in the figure below:



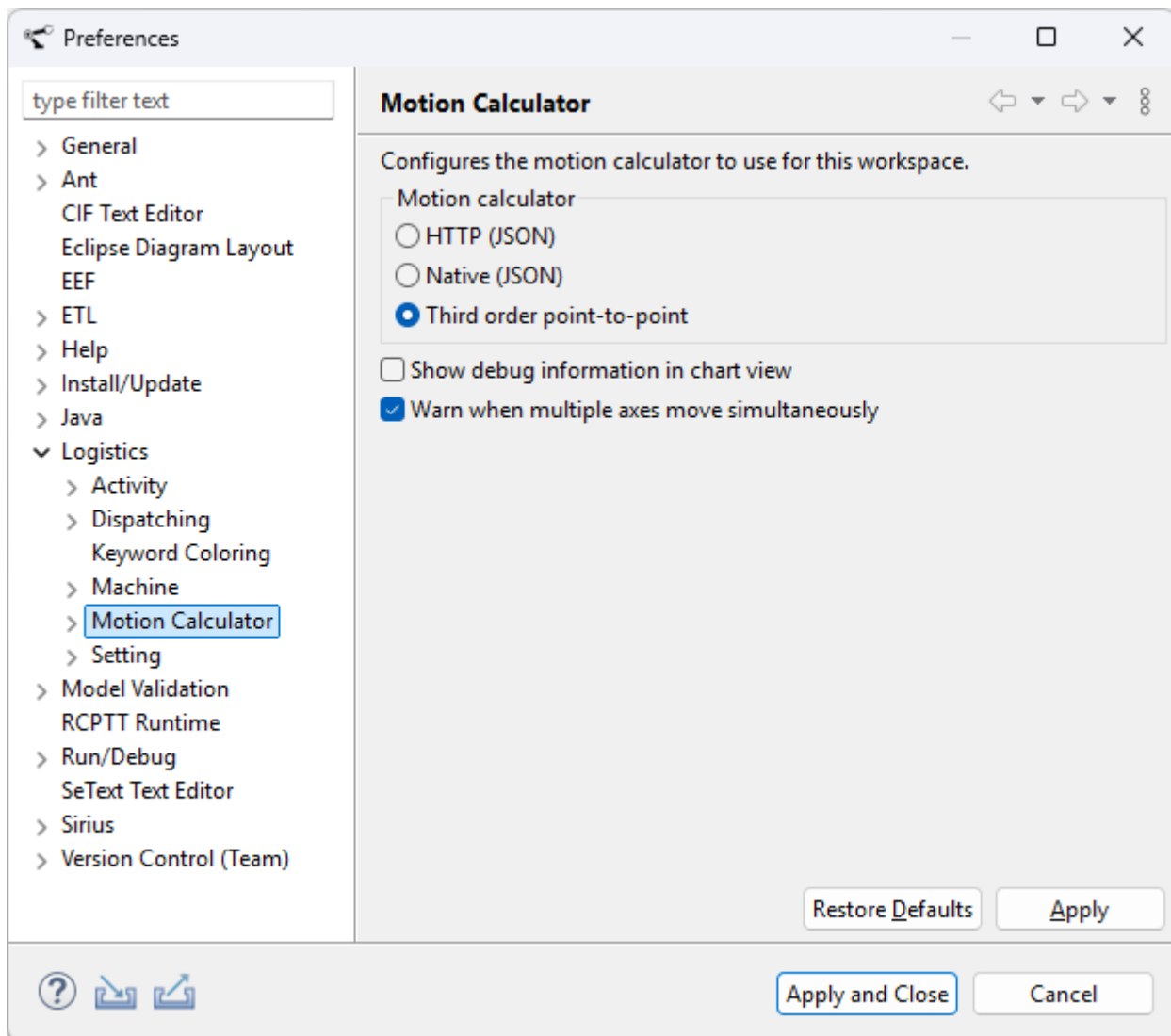
1.2. Motion Calculator

A motion calculator is used to calculate the time it takes for a move to complete. The default *Third order point-to-point* motion calculator is capable of calculating time for point-to-point moves with a 3rd order motion profile. It also supports passing moves, with the constraint that for all sub-moves the motion profile is the same.

Eclipse LSAT™ provides an extensible design for adding custom motion calculators. Creating a custom motion calculator requires knowledge about Eclipse plug-in development and Java. As such the instructions can be found in the [design documentation](#) of Eclipse LSAT™.

Eclipse LSAT™ also provides a simple JSON connector to add custom motion calculators written in other languages than Java. More details are provided in the [design documentation](#) of Eclipse LSAT™.

When a custom motion calculator is installed in the Eclipse LSAT™ environment, it can be selected via the **Window** › **Preferences** › **Logistics** › **Motion Calculator** menu.

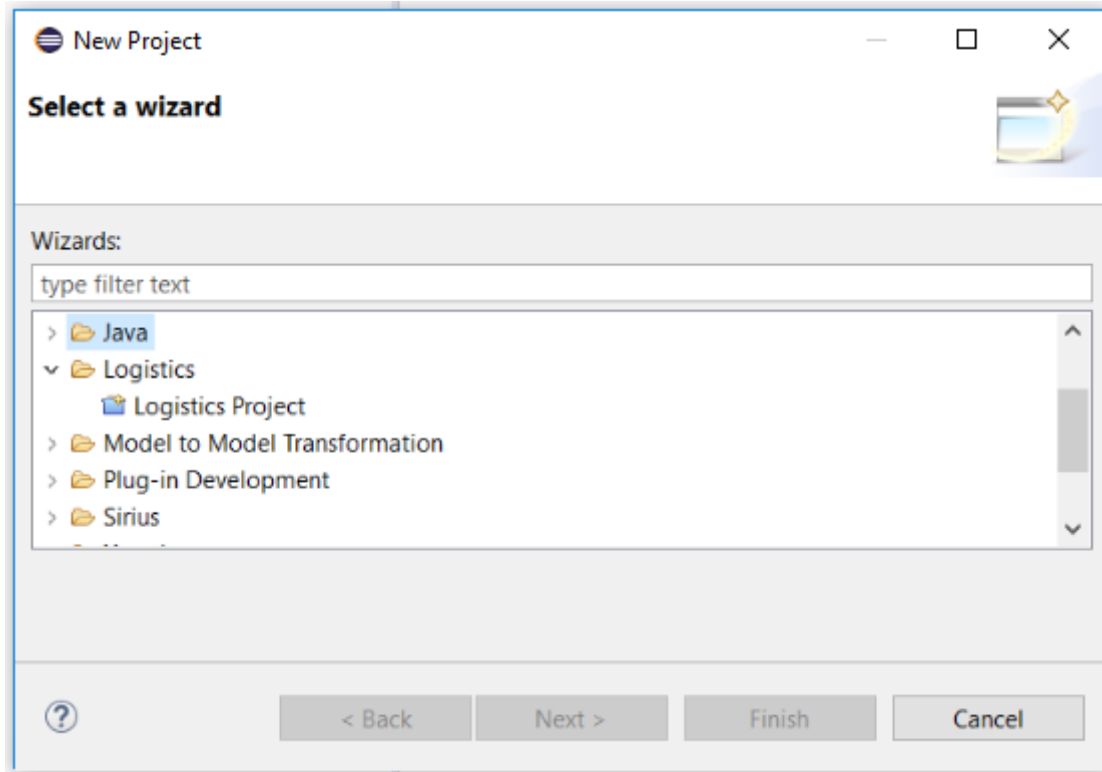


Chapter 2. Logistics Specification

2.1. Create New Project

You can create a new specification by using the **File > New** menu on the Workbench menu bar. Start by creating a logistics project as follows:

1. From the menu bar, select **File > New > Project...**



2. In the New Project wizard, select **Logistics > Logistics Project** then click Next.
3. In the Project name field, type your name as the name of your new project, for example "Tutorial".
4. Leave the box checked to use the default location for your new project. Click Finish when you are done.

The wizard creates a machine specification in the project and configures the project for graphical editing. It will open the [machine editor](#), allowing the user to add the resources and peripherals to the machine.

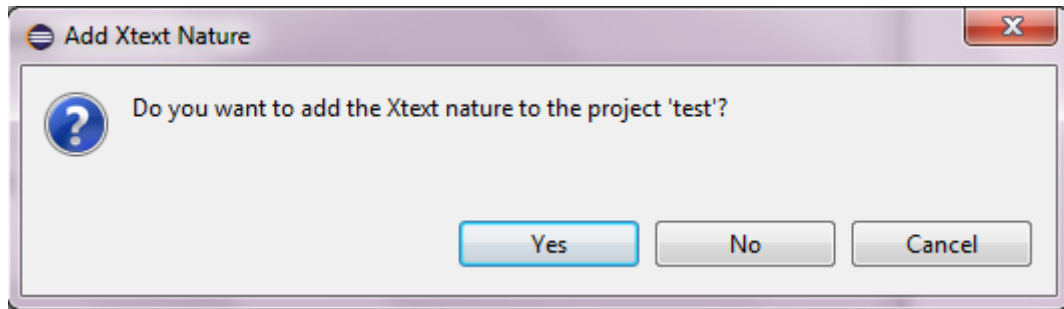
Within the project, the different types of specification can be created:

- machine specification
- settings specification
- activity specification
- activity dispatching specification

A new project always starts with machine specification, since it specifies the resources with their

peripherals.

If it is the first time a file is created in the project, a pop up dialog about adding Xtext nature to the project will appear. Click Yes to continue.



2.2. Create Machine Specification

Machine specification with file extension “machine” is used to specify the resources with their peripherals. This file is created automatically with the new project. It usually has the same name with the project, e.g. “tutorial.machine”.

A machine specification file can be opened to edit. Below are the basic concepts and small examples within a machine specification file.



`Ctrl` + `Space` can be used to see the hints for the next step in textual editors.

```
Machine Tutorial ①

PeripheralType Clamp { ②
  Actions {
    clamp
    unclamp
  }
}

PeripheralType Motor { ③
  Actions {
    apply_brake
  }
  SetPoints {
    X [m]
    Y
  }
  Axes {
    X [mm] moves X
    Y [mm] moves Y
  }
  Conversion "X=X/1000;Y=Y/1000" ④
}
```

```

Resource Robot1 { ⑤
  C: Clamp
  M: Motor {
    AxisPositions { ⑥
      X (Left, Right)
      Y (Above)
    }
    SymbolicPositions { ⑦
      Above_Left (X.Left, Y.Above)
      Above_Right (X.Right, Y.Above)
      Below_Left (X.Left)
      Below_Right (X.Right)
    }
    Profiles (normal, slow) ⑧
    Paths { ⑨
      Above_Right --> Below_Right [Y] profile slow
      Below_Right --> Above_Right profiles normal, slow ⑩
      Above_Left [X] <-> Above_Right profile slow
      FullMesh {
        profile normal
        Above_Left
        Above_Right
        Below_Left [Y]
      }
    }
  }
}

Resource Robot2 {
  C: Clamp

```

- ① **Machine** - the unique name for a machine
- ② **PeripheralType** (*unmovable*) – it only declares actions
- ③ **PeripheralType** (*movable*) - it may declare actions, but it also declares set points and axes.
 The **SetPoints** relate to the physical coordinate system of the peripheral on which the motion profile settings are applied.
 The **Axes** relate to the symbolic coordinate system on which the physical locations are applied.
 An axis specifies which set points change when its value changes. The unit is specified between the square brackets *[]* and defaults to **m**(eters).
- ④ **Conversion** - If the unit of an axis differs from the unit of its set points a conversion script is required. The conversion needs to convert a physical location in the symbolic axes coordinate system to a physical location in the physical set points coordinate system. For each axis a variable is available containing its coordinate value and for each set point a variable should be set containing the set point value.
 NOTE: The script language is JavaScript
- ⑤ **Resource** – it defines its peripherals. For every movable peripheral in a resource its symbolic positions and paths need to be specified.

- ⑥ **AxisPositions** (*optional*) - if a peripheral has multiple axes and the symbolic positions can be categorized in axis positions, these axis positions may be declared in this section. In this case only a physical location per axis position is required in the settings file for this axis instead of one location per symbolic position.
- ⑦ **SymbolicPositions** - declares the symbolic positions for this peripheral. Optionally a symbolic position may be expressed in its axes positions. If no axis position is specified (i.e. *Below_Left* and *Below_Right*) the symbolic position will automatically be added to the axis positions for the axis.
- ⑧ **Profiles** - declares the speed profiles to use for the paths.
- ⑨ **Paths** - declares which moves are allowed in the system. If a path is declared between two locations the move is allowed using the speed **profile** as specified by the path. For each end point of a path its settling axes can be specified as a comma separated list between square brackets []. Different types of paths are available:
unidirectional path - indicated by an unidirectional arrow -->, only a move from left to right is allowed
bidirectional path - indicated by a bidirectional arrow <->, both moves from left to right and right to left are allowed
full mesh - indicated by keyword **FullMesh**, all moves from a location to another location in the full mesh are allowed.
- ⑩ **Profiles** - You can specify more than one speed profile per path by means of the **profiles** keyword.

With the basic concepts as described above it is possible to make a full machine configuration.

2.2.1. Extended/Advanced Machine Specification

Machine configurations can be simplified, compressed or advanced using the concepts in below example:

```
import "tutorial.machine" ①

Machine Tutorial

Product sampleProduct { ②
    flipped : Boolean ③
}

PeripheralType LinearMotor {
    SetPoints {
        X
    }
    Axes {
        X [mm] moves X
    }
    Conversion "X=X/1000"
}
```

```

PeripheralType FlipperType {
    SetPoints {
        C
    }
    Axes {
        C [mm] moves C
    }
    Conversion "C=C"
}

Resource Robot(give,take) { ④
    C: Clamp
    M: LinearMotor {
        SymbolicPositions {
            Home
            Transfer
            Flip
        }
        Profiles (normal, slow)
        Paths {
            Home <-> Transfer profile normal
            Transfer <-> Flip profiles slow, normal
        }
    }
}

Resource Flipper {
    C: Clamp
    F: FlipperType {
        Profiles (normal)
        Distances { ⑤
            HalfRound
        }
    }
}

```

- ① **import** - machine files can be split up into multiple files.
- ② **ProductDefinition** - A product can be defined in the machine file. Its name can be specified alongside with all its properties.
- ③ **Properties** - Properties related to the product. These properties consist of a name and an associated type. Currently supported types are **Boolean**, **Integer** and **String**. These properties can be used and edited in the activity file.
- ④ **Resource items** - multiple identical resources can be declared with identifiers (name or number) between parentheses (Robot.*take*, Robot.*give* in the example).
- ⑤ **Distances** - Instead of positions, distances can be specified. Distances only describe a move of a certain distance where the location is unknown or not relevant. Typically used for circular, belt

or mill movements. Either (**AxisPositions SymbolicPositions and Paths**) or **Distances** must be specified. They are mutual exclusive.

2.3. Create Settings Specification

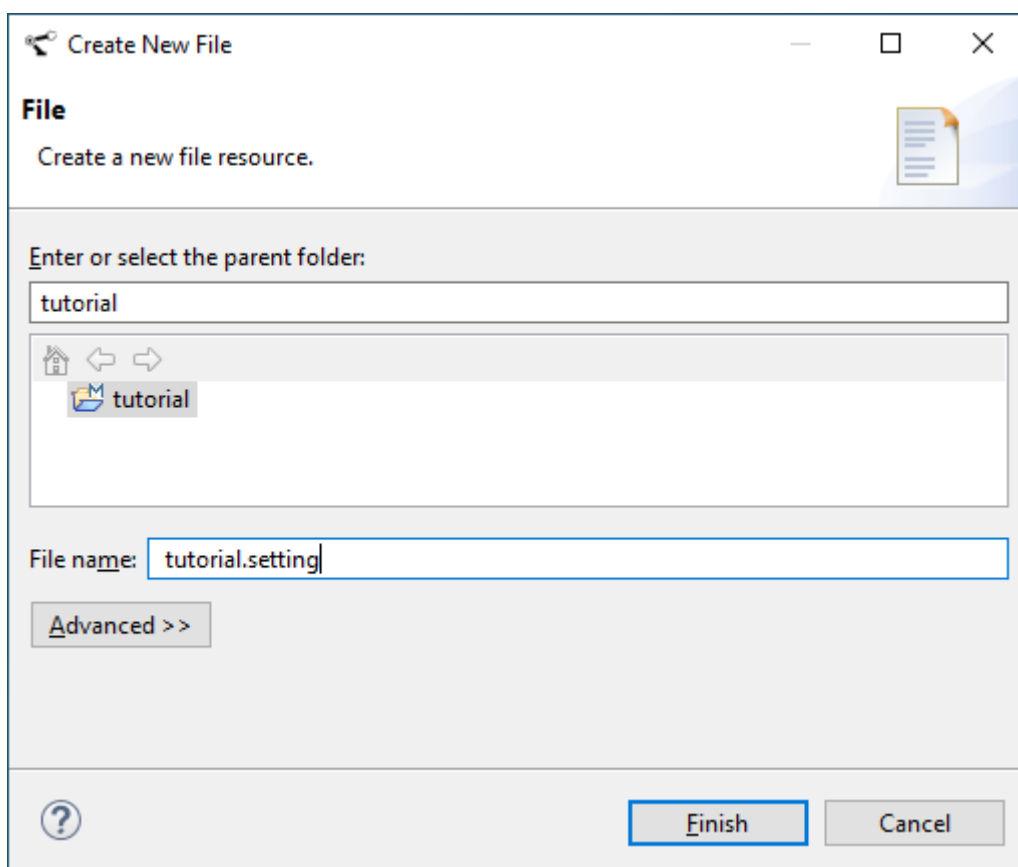
Settings specification with file extension “setting” is used to specify physical settings for peripherals or peripheral actions. Within the project, create a “.setting” file by using the new file wizard.



Multiple setting files are allowed in a project. They can be used to explore different system configurations



If another file is selected while running the new file wizard, this file will be automatically imported



A setting file is opened. Below are the basic concepts in a settings file, illustrated with small examples.



`Ctrl` + `Space` can be used to see the hints for the next step in textual editors.

```
import "tutorial.machine" ①

Robot1.M { ②
  Timings { ③
    apply_brake = 100e-3
  }
}
```

```

Axis X {
  Profiles { ④
    slow (V = 0.5, A = 0.5, J = 0.5, S = 0.1)
    normal (V = 1, A = 1, J = 1)
  }
  Positions { ⑤
    Left (min = -100, max = -200, default = -150)
    Right = 300.0
  }
}
Axis Y {
  Profiles {
    slow (V = 0.5, A = 0.5, J = 0.5, S = 0.1)
    normal (V = 1.0, A = 1.0, J = 1.0, S = 0.1)
  }
  Positions {
    Above = 150.0
    Below_Left = -160.0
    Below_Right = -150.0
  }
}
MoveAdjustments { ⑥
  // add 'jitter' to profile slow:
  time = Normal(mean = time, sd = time * 0.02) profile slow
  // add 'jitter' to any other profile:
  time = Normal(mean = time, sd = time * 0.03)
  // example how to add 1 second to any other profile:
  // time = time + 1
}
}

Robot1.C {
  Timings { ③
    clamp = Normal(mean = 1.1, sd = 0.1)
    unclamp = 0.9
  }
}

```

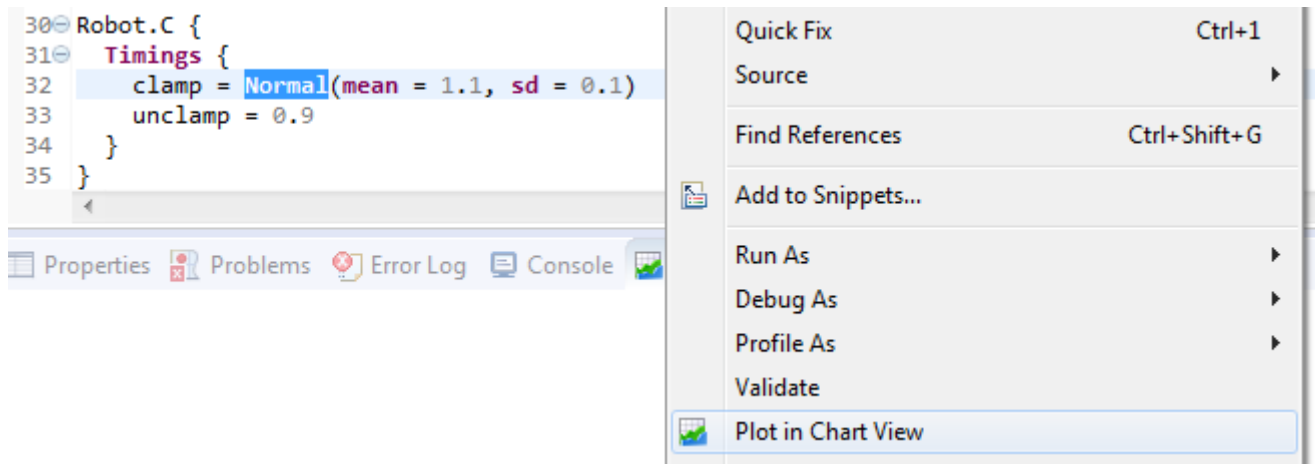
- ① **import** - first step in a setting specification is to import a machine or other setting file. After that, it is allowed to specify properties for peripherals or actions.
- ② **Robot.M** - for a movable peripheral, it is required to set the motion profiles to its profiles and physical relative locations to its symbolic positions. These properties are always set per axis.
- ③ **Timings** - if a peripheral contains actions, a timing must be specified. Any *Expression* may be used.
- ④ **Profiles** - motion profiles need to be assigned to each profile individually. For the built-in motion calculator they can be either Third-order profile (**VAJ**) or Second-order profile (**VA**). Optionally a settling time (**S**) can be specified per profile.
- ⑤ **Positions** - physical relative locations can be specified as a set of maximum, minimal and

default values or one fixed value.

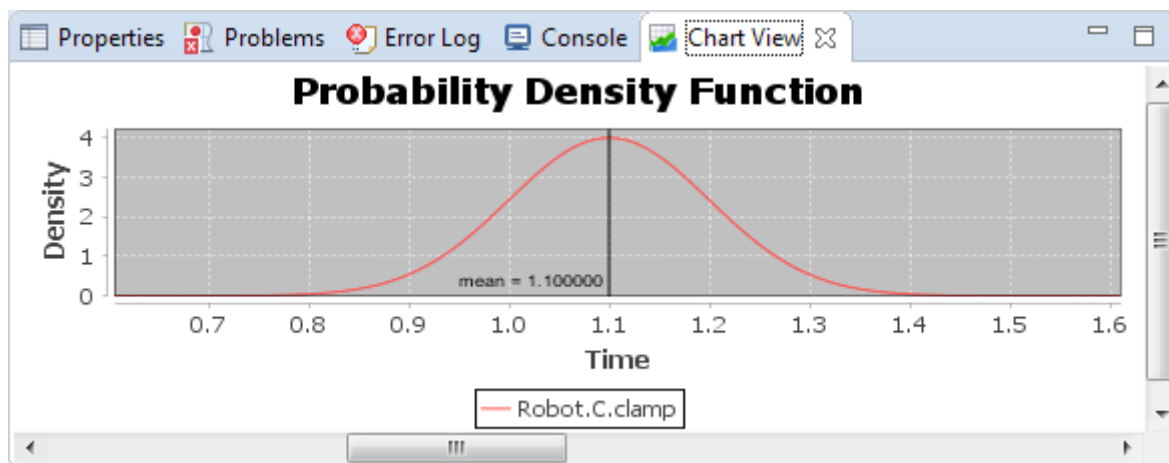
- ⑥ **MoveAdjustments** - [optional] A correction of the timings calculated by the motion calculator. To be used for instance to specify any Jitter, ramp up/down or anything else. Any *Expression* may be used. The calculated time is referred to as a **time Expression**.

2.3.1. Plotting a timing distribution

To gain insight in the probability density for timings in a distribution, the probability density factor (PDF) can be plotted in the Chart View. To do this, please select the distribution keyword of the requested distribution and select the 'Plot in Chart View' item from the context menu, as illustrated in the next figures



The chart view also shows the *mean* marker, this is the value used for scheduling.



2.3.2. Extended/Advanced Settings Specification

Setting files can be advanced using Expressions and/or combined Resource items:

```
import "tutorial_ext.machine"  
import "tutorial.setting"  
  
// area settings.  
val width = 300 ①  
val height = 300
```

```

val right = width/2 ②
val left = width/-2
val homeLeft = left - 10
val homeRight = right + 10
val center = left + (right - left)/2
// timings
val breakTiming = 100e-3

//profiles (artificial)
val vRSlow = 0.5
val aRSlow = 0.5
val jRSlow = 0.5
val sRSlow = 0.1
val vRNormal = vRSlow*2
val aRNormal = vRSlow*2
val jRNormal = vRSlow*2
val sRNormal = 0.1
val clampLow = 1
val clampSD = 0.1

Robot.give.M { ③
  Axis X {
    Profiles {
      slow (V = vRSlow, A = aRSlow, J = jRSlow, S = sRSlow) ④
      normal (V = vRNormal, A = aRNormal, J = jRNormal)
    }
    Positions {
      Transfer = left
      Home = homeLeft
      Flip = center
    }
  }
}

Robot.take.M { ③
  Axis X {
    Profiles {
      slow (V = vRSlow, A = aRSlow, J = jRSlow, S = sRSlow)
      normal (V = vRNormal, A = aRNormal, J = jRNormal)
    }
    Positions {
      Transfer = right
      Home = homeRight
      Flip = center
    }
  }
}

Robot.C { ⑤
  Timings {
    clamp = Normal(mean = clampLow+clampSD, sd = clampSD) ⑥
  }
}

```

```

    unclamp = 0.9
  }
}

Flipper.F {
  Axis C {
    Profiles {
      normal (V = 1, A = 1, J = 2)
    }
    Distances { ⑦
      HalfRound = 0.5
    }
  }
}

Flipper.C {
  Timings {
    clamp = 0.5
    unclamp = 0.9
  }
}

```

- ① **Expressions** - instead of magic numbers Expressions can be used. Re-usable Expressions can be declared anywhere in the settings file. Expressions can also be Distributions for example `val normalDist(mean = 0, sd = 0.1)`
- ② **Calculations** - simple calculations using `+` `-` `*` `/` `%` `^` and parentheses are supported. Also trigonometry functions like **Sin**, **Cos**, **Tan**, **Asin**, **Acos**, **Atan**, **Atan2** and the constant **Pi** are supported, as well as the functions **Abs**, **Exp**, **Log**, **Min**, **Max**, **Round**, **Floor**, **Ceil** and **Sqrt**. The arguments of the calculations can also be Distributions.
- ③ **Different settings for Resource items** - Place the identifier as specified in the machine file between the Resource and the Peripheral to make specific settings for a Resource item (In this example: Robot.*give*.M and Robot.*take*.M).
- ④ **Usage** - Expressions can be used to calculate the actual properties.
- ⑤ **Resource items** - multiple resources with identical Peripheral settings can be specified using the Resource name.
- ⑥ **Expressions in assignments** - Expressions calculations can directly be used in assignments.
- ⑦ **Distances** - when using distances the Distance must be specified as described.



Hovering over an Expression or an assignment yields the calculated result in a pop-up window.



For a Peripheral within a Resource, settings must be specified for either the Resource or **all** Resource items. An error will be raised if a combination is made.

Explore different settings.

Similar to machine files, setting files can also be split up into multiple files. **import** of machine and other setting files is supported. In [Create Activity Dispatching Specification](#) is described how to choose different setting files.

2.4. Create Activity Specification

Activity specification with file extension “activity” is used to specify activities of a machine. Within the project, create an “.activity” file by using the new file wizard.



If another file is selected while running the new file wizard, this file will be automatically imported

An activity specification file is opened. Below are the basic concepts and small examples within a activity specification file.



`Ctrl` + `Space` can be used to see the hints for the next step in textual editors.

```
import "tutorial.machine" ①

activity MoveRobot { ②
  prerequisites { ③
    Robot1.M at Above_Right
  }
  actions { ④
    C: claim Robot1
    R: release Robot1
    A1: move Robot1.M to Below_Right with speed profile slow
    A2: Robot1.C.unclamp
    // Schedule A3 to end together with A4 (As Late As Possible)
    A3: Robot1.C.clamp ALAP ⑤
    A4: move Robot1.M to Above_Right with speed profile normal
  }
  action flow { ⑥
    C -> A1 -> A2 -> |S1 -> A3 -> |S2 -> R
                    |S1 -> A4 -> |S2 ⑦
  }
}
```

- ① **import** - first step in an activity specification is to import a machine specification file.
- ② **activity** - specify activities by defining individual actions and dependencies among these actions.
- ③ **prerequisites** - specify the initial location for all movable peripherals for this activity.
- ④ **actions** - there are four types of actions:
 1. *Resource claiming* – action name, **claim** keyword and resource.
A resource can be passively claimed to ensure that no actions are performed by any of its

peripherals until the resource is released again. To passively claim a resource, the **claim** keyword can be preceded by the **passive** keyword: **passive claim** <resource>. If more activities claim a resource passively, they potentially can run in parallel.

2. *Resource releasing* – action name, **release** keyword and resource
3. *Peripheral action* – action name, resource, peripheral and peripheral action
4. *Move* - action name, **move** keyword, resource, peripheral, symbolic position destination and speed profile.

The default *Move* action is point-to-point (stop at target position). However, it can be changed to **passing** (positions) or **continuing** (distances). By using the keyword, you specify that the action should not stop at target. Another name for **passing/continuing** is event-on-the-fly. This feature is used to concatenate *Move* actions and the user needs to explicitly enable it. Here is an example declaration of a passing move:

A1: move Drill.ZR passing UP with speed profile normal ALAP

An example for distance moves can be found in [Extended/Advanced Activity Specification](#)

The images below illustrate the difference between two types of *Move* actions.

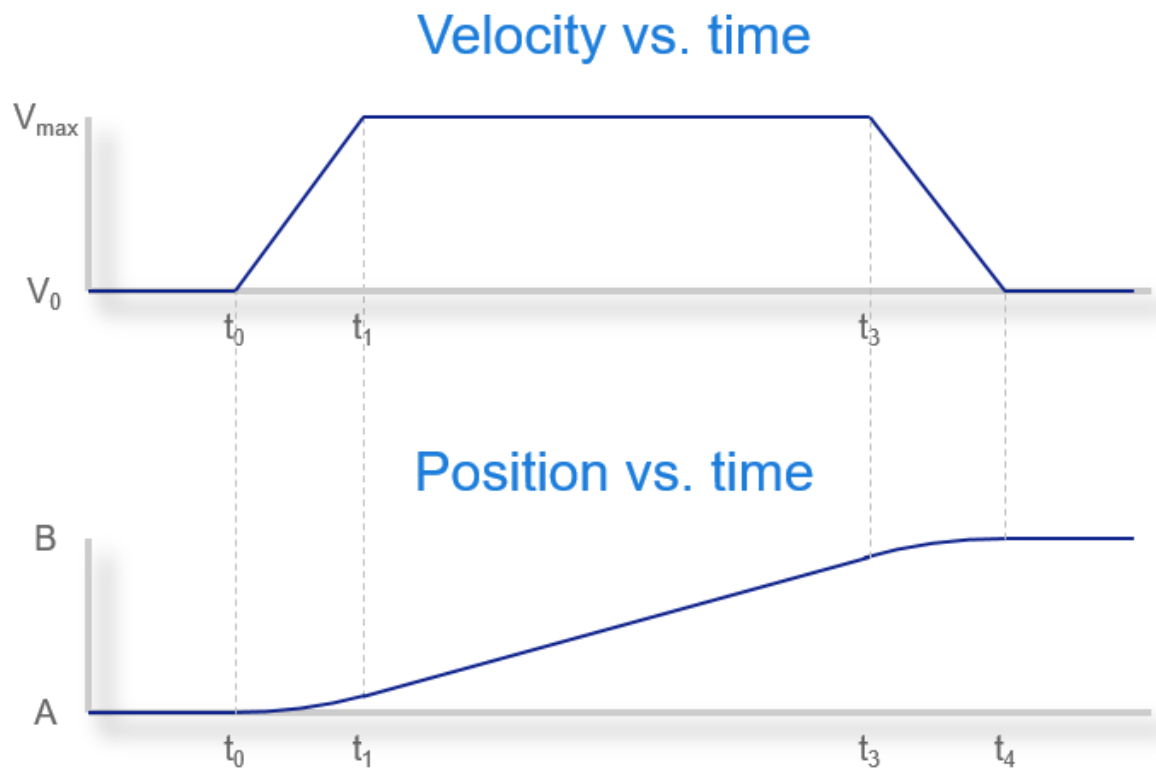


Figure 1. Movement point-to-point.

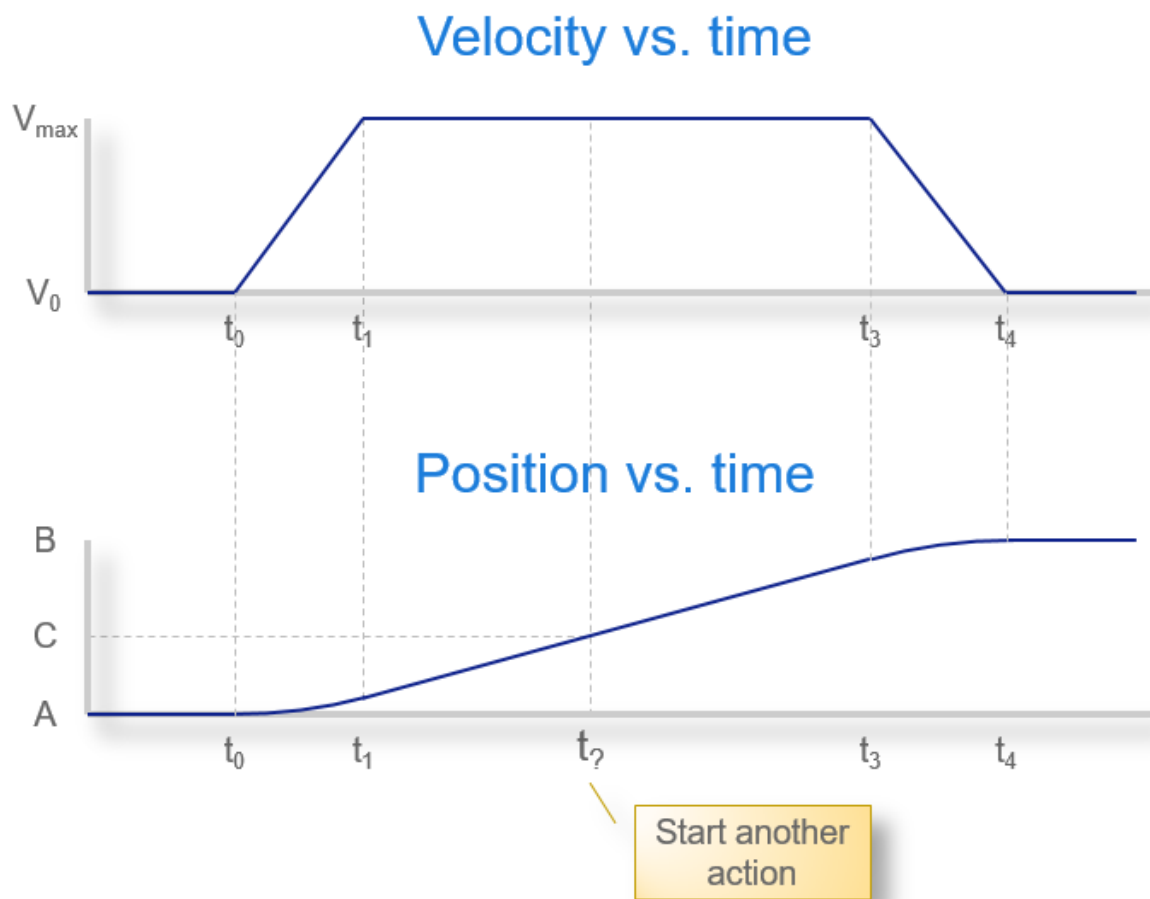


Figure 2. Movement on-the-fly event.

- ⑤ **ALAP / ASAP** - Each action has *scheduling type*. There are two types available: As Soon As Possible (ASAP) and As Late As Possible (ALAP). If no value is specified at the end of the action, then the scheduling type is **ASAP** by default. To change it to **ALAP** you can add the **ALAP** keyword at the end of the action. For more information, see section [Constraints for ALAP and ASAP scheduling actions](#).

Resource claiming – **ALAP** by default, cannot be changed.

Resource releasing – **ASAP** by default, cannot be changed.

Peripheral action – **ASAP** by default, can be changed to **ALAP**.

Move - **ASAP** by default, can be changed to **ALAP**.

- ⑥ **action flow** - uses arrows to specify action flow. $A1 \rightarrow A2$ represents A2 starts when A1 finishes.
- ⑦ **synchar** - used to specify synchronization, for instance $A2 \rightarrow |S1 \rightarrow A3$ and $|S1 \rightarrow A4$ represents A3 and A4 start simultaneously after A2 finishes.

2.4.1. Extended/Advanced Activity Specification

Distance moves and resource items can be used in activities, as specified below:

```
import "tutorial_ext.machine"

activity MoveAndFlip {
  prerequisites {
    Robot.give.M at Transfer
```

```

    Robot.take.M at Transfer
}
actions {
    RGClaim: claim Robot.give ①
    RGRelease: release Robot.give
    FClaim: claim Flipper
    FRelease: release Flipper
    RTClaim: claim Robot.take
    RTRelease: release Robot.take
    RGMoveFlip: move Robot.give.M to Flip with speed profile normal
    RGMoveTransfer: move Robot.give.M to Transfer with speed profile slow
    RTMoveFlip: move Robot.take.M to Flip with speed profile normal
    RTMoveTransfer: move Robot.take.M to Transfer with speed profile slow
    RTClamp: Robot.take.C.clamp in transfer2②
    RGUnclamp: Robot.give.C.unclamp out transfer1 ③
    RGClamp: Robot.give.C.clamp entry @start (flipped: False) ④
    RTUnclamp: Robot.take.C.unclamp exit ⑤
    FUnclamp: Flipper.C.unclamp out transfer2 @start(flipped : True) ⑥
    FClamp: Flipper.C.clamp in transfer1 ⑦
    FFlip: move Flipper.F for HalfRound with speed profile normal ⑧
}
action flow {
    RGClaim -> RGClamp -> RGMoveFlip -> FClaim->FClamp -> RGUnclamp -> |S1 ->FFlip ->
|S2
    RTClaim -> RTMoveFlip -> |S2
    |S1 -> RGMoveTransfer -> RGRelease
    |S2 -> RTClamp -> FUnclamp -> FRelease-> RTMoveTransfer -> RTUnclamp -> RTRelease
}
}

activity RobotMove(robot: Robot, start, end : SymbolicPosition) { ⑨
    prerequisites {
        robot.M at start
    }
    actions {
        Claim: claim robot
        Release: release robot
        MoveTransfer: move robot.M to end with speed profile normal
    }
    action flow {
        Claim -> MoveTransfer -> Release
    }
}

```

- ① **Resource item** - In an activity a resource item can be specified by the resource name followed by the identifier (Robot.give in the example).
- ② **in** - Denotes the product flow, stating that the peripheral receives a product with this action. Should always be coupled with an **out** action.

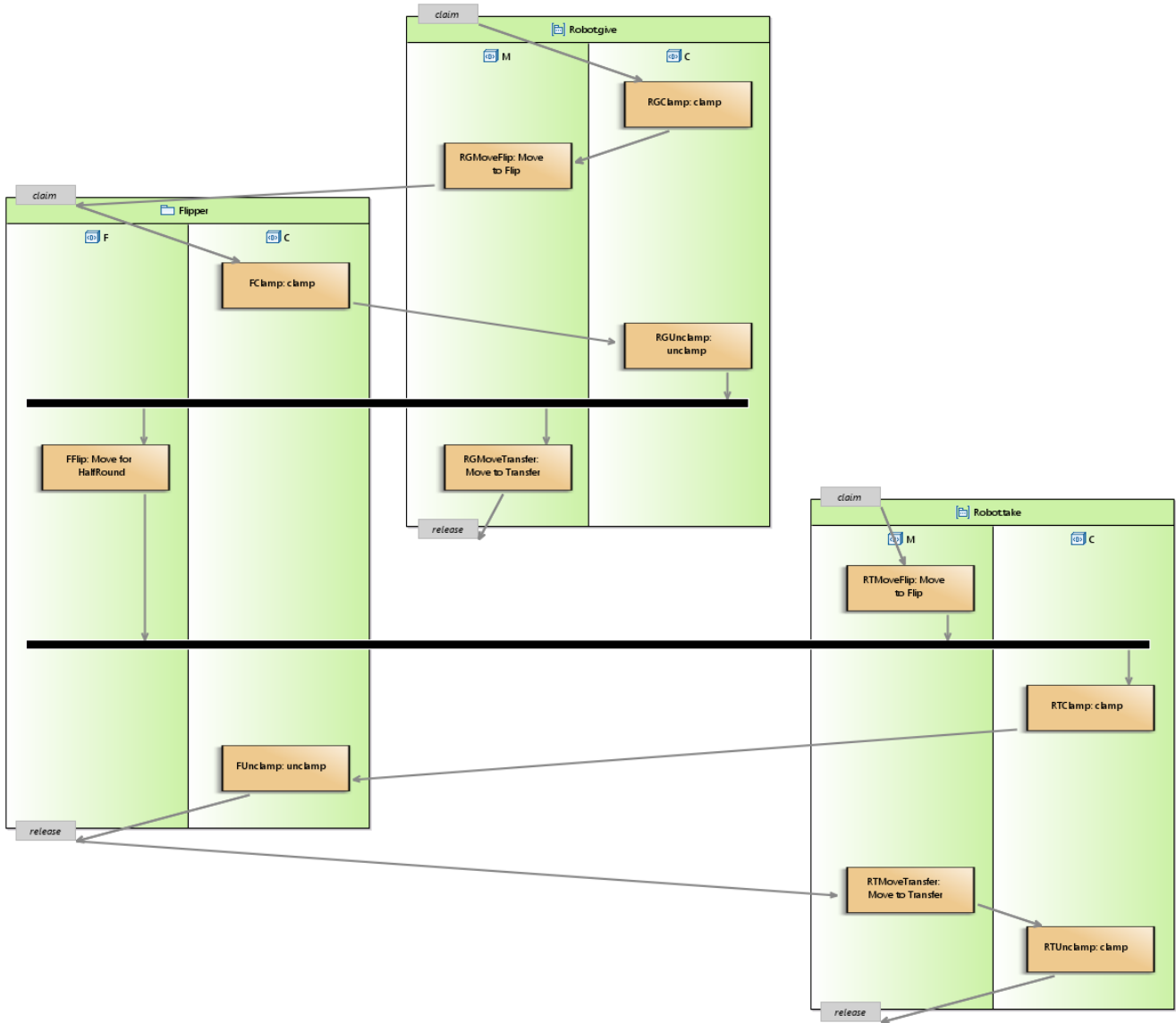
- ③ **out** - Denotes the product flow, stating that the peripheral provides a product with this action. Should always be coupled with an **in** action.
- ④ **entry** - Product enters the system. Properties defined in the ProductDefinition in machine file can be initialized here. For example, the product property **flipped** starts with the value **False**.
- ⑤ **exit** - Product that is currently active exits the system. This ensures that it cannot be referred to in any following actions.
- ⑥ **@start** - Updates the value (at start,) of a specified product property with the newly given value.
- ⑦ **transfer1** - Transfers between 'in' and 'out' can be explicitly paired. This is only needed if there is ambiguity in the destination of a product. The product will be transferred to/from the action with a matching slot name. If no specific name is specified the product is transferred using the **default** slot.
- ⑧ **Resource** - An activity can also refer to a resource, without specifying the specific resource item. The advantage of this is that the activity specification can be reused and instantiated for the specific resource items. The specific resource item is assigned in the activity instance in the activity dispatching specification. **This option is now obsolete and must be replaced with Parameters.**
- ⑨ **Parameters** - For reuse and conciseness activities can have parameters. Their type can be a concrete global Resource (with resource items) or Product definition and types like SymbolicPosition, Action, Profile, Resource and Event. The declared parameters have to be provided in the dispatching files. See also [Extended/Advanced Activity Dispatching Specification](#) for usage and [Parameterized Activities Details](#) for details.



The shown activity is an activity that brings a product from *left* to *right* using 2 robots. The product is flipped in the center. Graphically it looks as follows:



an **out** transfer is always from the same peripheral. (only) If there are multiple products on one peripheral, the first product that has not yet passed the transfer position is chosen.



2.4.2. Constraints for ALAP and ASAP scheduling actions

There are a number of validation rules implemented in Eclipse LSAT™ for the different scheduling algorithms. In general you have to keep two things in mind:

1. The [scheduler algorithm](#) prefers **ASAP** scheduling over **ALAP** scheduling.
2. A concatenated move with events on-the-fly may be delayed as a whole, but not in between its events.

Scenario 1

Consider action A1 in the [ALAP scenario 1](#) with scheduling type **ALAP** and a sole successor (A2), which is not a synbar. In this case the **ALAP** keyword does not have any effect due to the **ASAP** (default) scheduling of action A2.

Eclipse will show you the following warning:



The ALAP keyword does not have any effect for action A1.

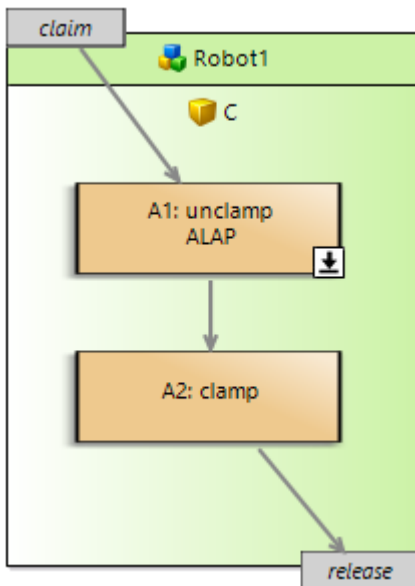


Figure 3. ALAP scenario 1

Scenario 2

Consider move A1 in [ALAP scenario 2](#) which is **passing**. As its successor is a synchar, and this synchar has more than one predecessor, then **move** A1 must be **ALAP**. As moves A1 and A3 should be concatenated, there cannot be any delay between them. As action A2 might cause such a delay, move A1 needs to be scheduled ALAP to guarantee concatenation.

Eclipse will show you the following error:



Move A1 must be set to ALAP to guarantee concatenation with successor move A3.

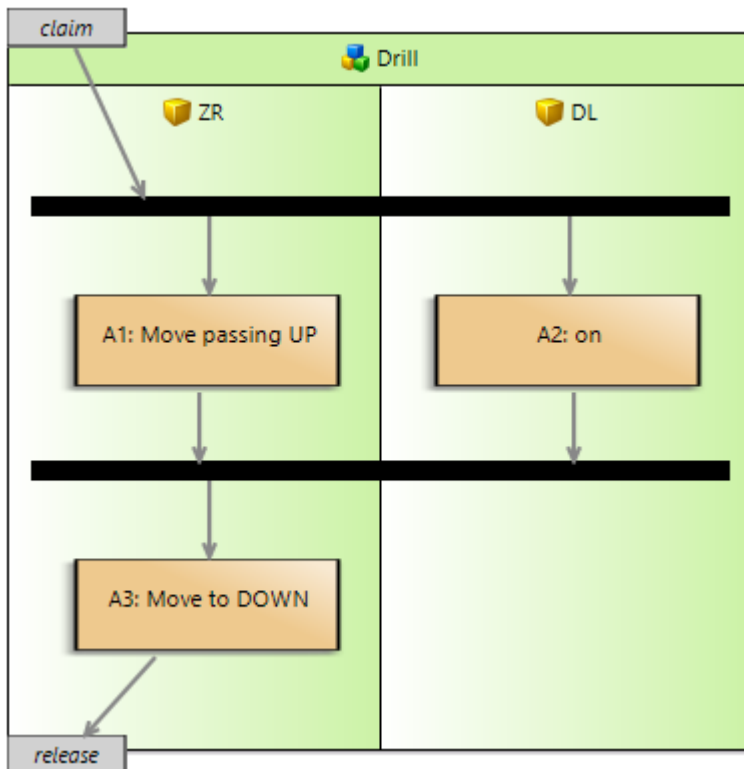


Figure 4. ALAP scenario 2

Scenario 3

Consider move A3 in [ALAP scenario 3](#) which is a continued move of A1. Move A1 continues into move A3 by means of the **passing** keyword and therefore move A3 cannot be delayed. As the predecessor of move A3 is a synchbar, move A3 must be **ASAP** to guarantee concatenation.

Eclipse will show you the following error:



Move A3 must be set to ASAP to guarantee concatenation with predecessor move A1.

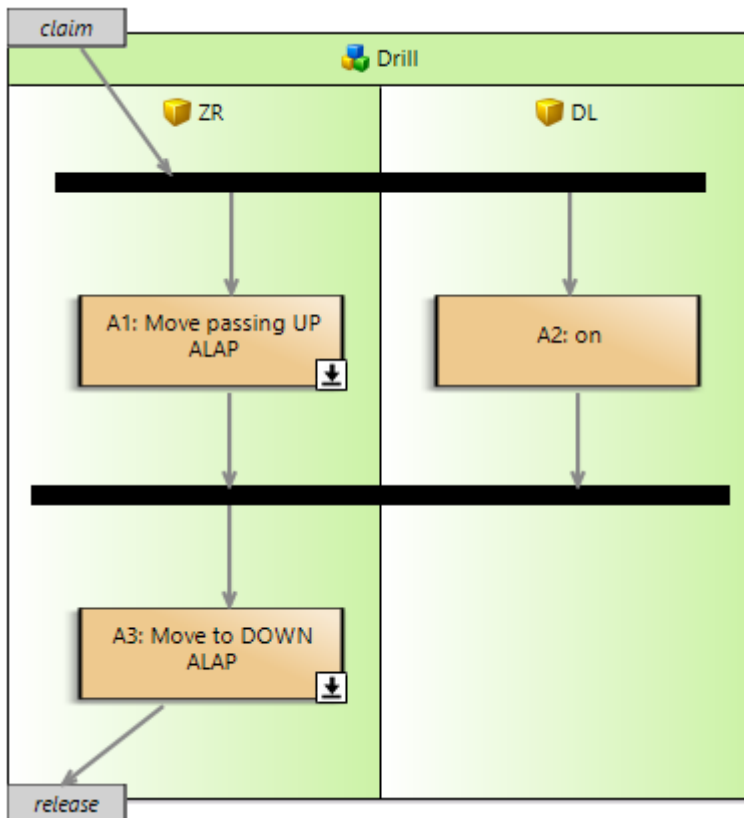


Figure 5. ALAP scenario 3

Scenario 4

Consider move M3 in [ALAP scenario 4](#) which is a continued move of M1 and M2.

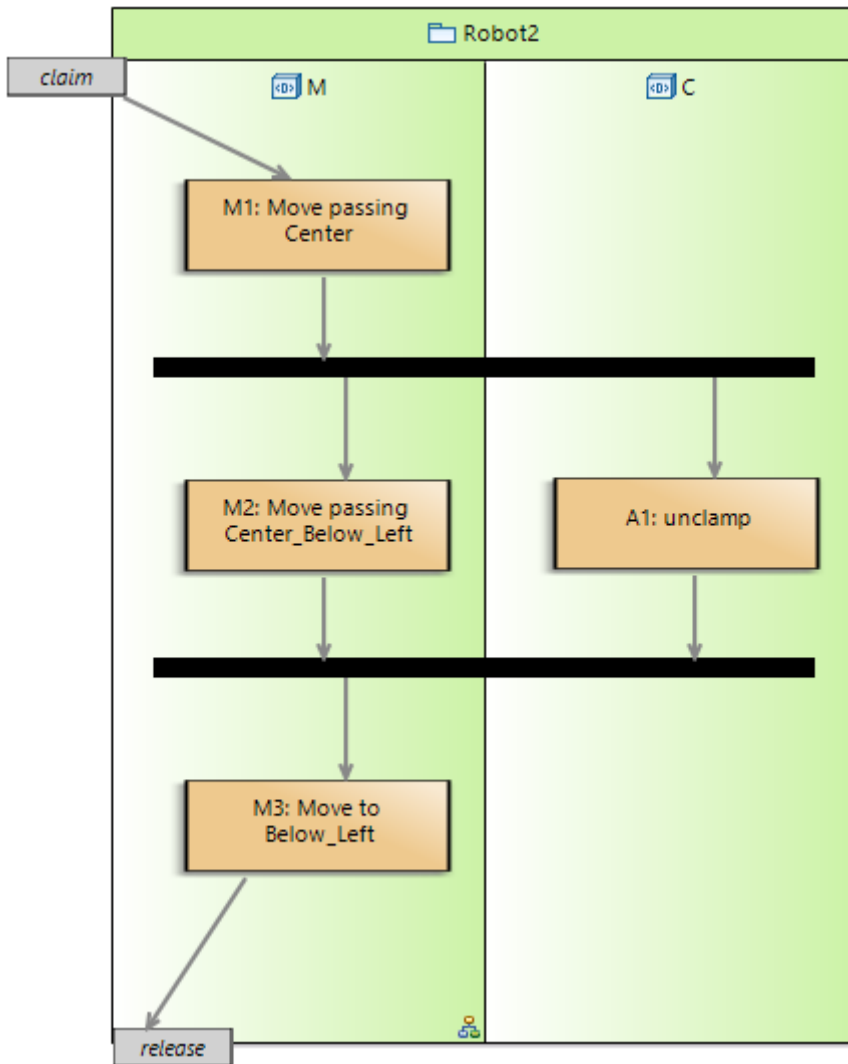


Figure 6. ALAP scenario 4

Move M2 finally continues into move M3 by means of the **passing** keyword and therefore move M3 cannot be delayed. Because action A1 is parallel to M2 this cannot be guaranteed hence to following warning will be shown:



Parallel action 'A1' might interrupt concatenated move 'M3'.

If after a Timing analysis it appears that M3 is indeed delayed the following error will be shown:

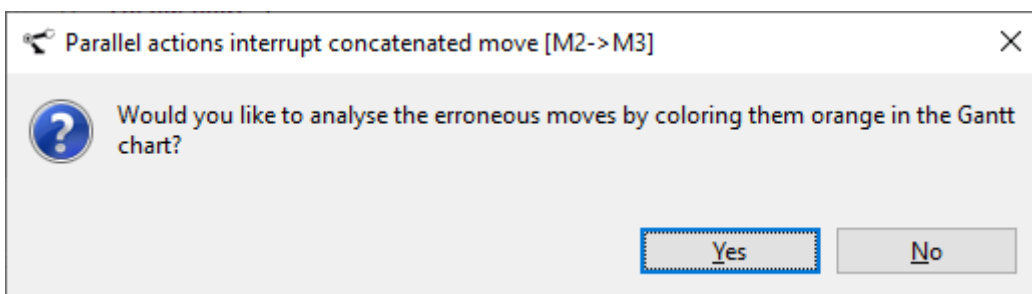


Figure 7. ALAP scenario 4 Error

In the [ALAP scenario 4 Gantt chart](#) the erroneous move is colored orange.

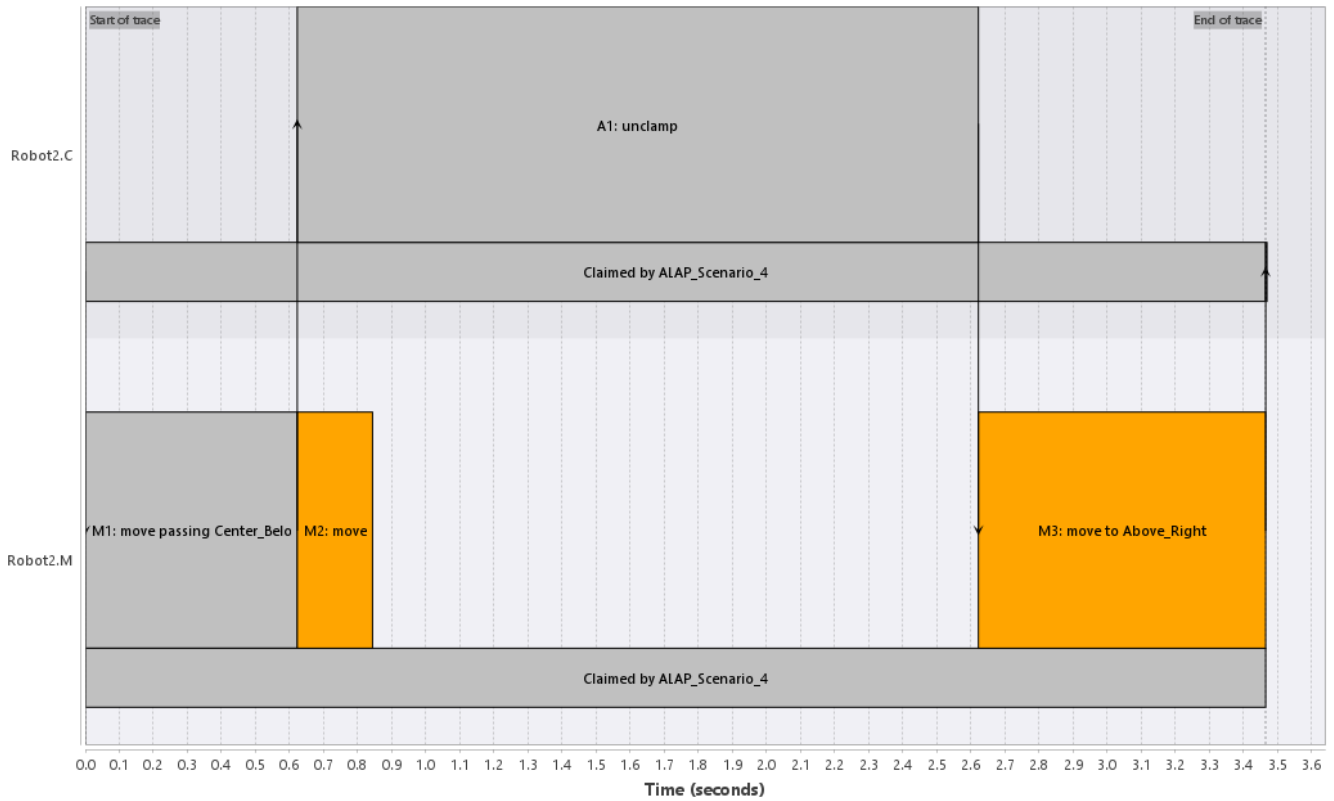


Figure 8. ALAP scenario 4 Gantt chart

JIT and time-constraints

Next to defining actions as **ASAP** or **ALAP**, JIT (Just-In-Time) and time constraints can be defined between two actions: *source* and *target*. These two actions do not need to be consecutive, there can be other actions in between.

A JIT constraint implies that the *source* and *target* action (and all possible actions in between) are scheduled such that there is no idle time in between the *source* and the *target* action. This could mean the *source* action is delayed until the JIT constraint can be met, which can also cause other actions depending on *source* to be scheduled later. This is different than scheduling an action **ALAP**, which will only delay the action itself but not other actions.

A time constraint defines a lower- and upper-bound between the end of the *source* action and the start of the *target* action. The *source* and *target* actions will be scheduled in such a way that the start of the *target* action is between the lower- and upper-bound later than the end of the *source* action, independent of any intermediate actions.

When JIT or time-constraints cannot be fulfilled, LSAT will give an error showing which constraints could not be met.

JIT and time constraints can be defined in either the activity specification or the activity dispatching specification. In the activity specification, constraints can either be added directly in the **action flow** for constraints between consecutive actions, or in the **constraints** section. Both options are shown below:

```
action flow {
  claim -> A -> B -jit-> C -> D -[2,4] -> E -> release
```

```

}

constraints {
  A -jit-> C
  C -[5,8]-> E
}

```

JIT and time constraints can also be defined in the activity dispatching specification (see [Create Activity Dispatching Specification](#)), to either specify constraints between actions in different activities, or to give different constraints for different instantiations of an activity. Constraints can either be specified in a **constraints** section inside an activity dispatching, for constraints within one activity dispatching, or in a **constraints** section after all activity dispatchings for constraints. Both options are shown below:

```

activities init {
  Act1: Activity1
  Act2: Activity2

  constraints {
    Act1.D -jit-> Act2.A
    Act2.A -[1,10]-> Act2.B
  }
}

activities loop repeat : [1..5] {
  Act2: Activity2

  # JIT constraint between A and B for every repeat
  constraints {
    Act2.A -jit-> Act2.B
  }
}

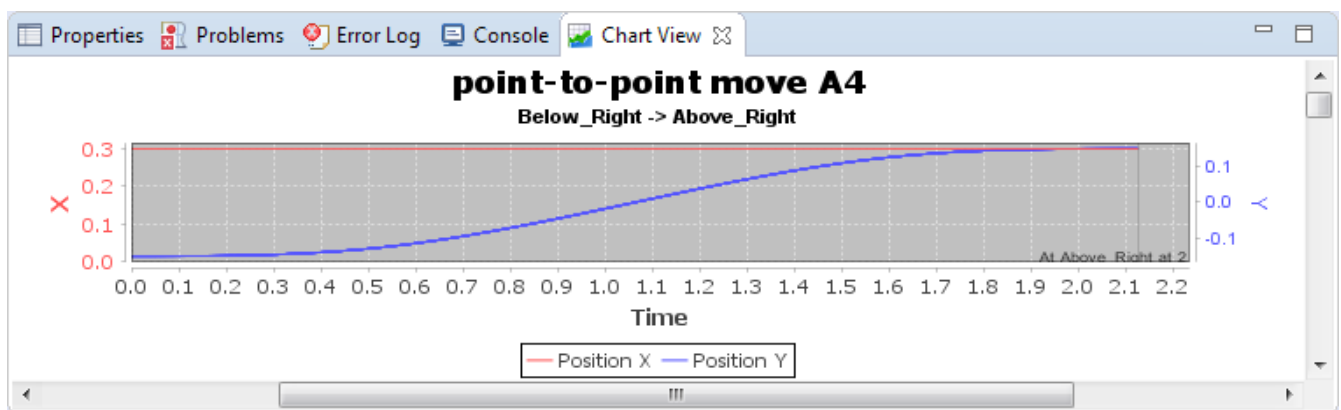
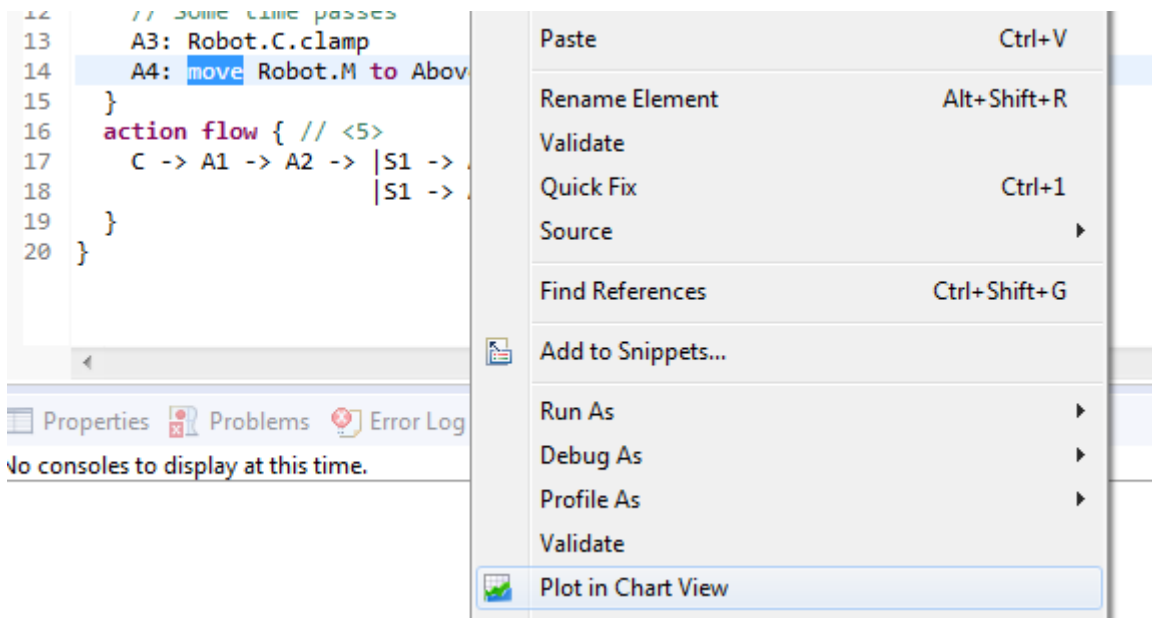
# time constraint between Act2.E in init and the first Act2.E in loop
constraints {
  init.Act2.E -[0,20]-> loop.Act2.E
}

```

Note: a regular arrow in constraints sections (->) is a short-hand notation for a time constraint with bounds $[0, \infty]$ (-[0,inf]->).

2.4.3. Plotting a move

To gain insight in the motion path of a move, the motion path can be plotted in the Chart View. To do this please select the **move** keyword of the requested move and select the 'Plot in Chart View' item from the context menu, as illustrated in the next figures



By right-clicking on the plot, a menu can be opened which allows selecting a plot. For the *Third order point-to-point* motion calculator the available plots are: position, velocity, acceleration and jerk.

2.5. Create Activity Dispatching Specification

Activity dispatching specification with file extension “dispatching” is used to specify a sequence of activities to schedule. For more information about scheduling, see section [Schedule Activities](#). Within the project, create a “.dispatching” file by using the new file wizard.

An activity dispatching file is opened. Below are the basic concepts and small examples within a activity dispatching file.



Ctrl + Space can be used to see the hints for the next step in textual editors.

```
import "tutorial.activity" ①

throughput {
    iterations: 1 ②
    iterations.Robot1: 2
}
```

```

activities { ③
  MoveRobot ("First product")
}

activities {
  offset: 6.00 ④
  MoveRobot ("Second product")
}

```

- ① **import** - first step in an activity dispatching is to import an activity specification file.
- ② **iterations** - specifies the default number of iterations - for all resources - or the number of iterations for a specific resource (see next line). When omitting the **throughput** section the default number of iterations will be set to 1. For more information about **iterations** see section [Specifying the number of iterations](#).
- ③ **activities** - within an activity sequence one or more activities can be included. These activities will be scheduled as parallel as possible adhering to the claim/release strategy. If activities cannot be scheduled in parallel the order of this section is applied.
Optionally a description can be added to the activity
- ④ **offset** - optionally an offset can be defined for an activity sequence. The scheduler will ensure that these activities will not be started before the offset time.

2.5.1. Extended/Advanced Activity Dispatching Specification

```

import "tutorial_ext.activity" ①

activities rampUp { ②
  RobotMove[give, Home, Transfer] ③
  RobotMove[take, Home, Transfer]
}

activities steadyFlow(anAttribute:1) ④
  repeat coin: [2,4] { ⑤
    yield: 1 ⑥
    MoveAndFlip(step:1,scenario:"normal") ⑦
    MoveAndFlip(step:2,scenario:"normal")
  }

/** steady flow expands to:
activities steadyFlowExpanded(anAttribute: 1) { ⑧
  yield: 2
  MoveAndFlip(step: 1, scenario: "normal", phase: steadyFlow, coin: 2)
  MoveAndFlip(step: 2, scenario: "normal", phase: steadyFlow, coin: 2)
  MoveAndFlip(step: 1, scenario: "normal", phase: steadyFlow, coin: 4)
  MoveAndFlip(step: 2, scenario: "normal", phase: steadyFlow, coin: 4)
}
*/

activities tearDown { ②
  RobotMove[give, Transfer, Home ]
}

```

```
RobotMove[robot=take, start=Transfer, end=Home ] ⑨  
}
```

- ① **import** - import one or more activity files.
- ② **activity phase** - a phase name may be specified to indicate a specific phase like start-up, steady-flow, or tear-down.
- ③ **parameters** - for activities that have parameters or their specific resource item not yet specified the parameters must be specified (For example: The RobotMove activity requires 3 parameters).
- ④ **user attributes** - user info can be added for an activity. This information will be passed to the trace file see [Schedule Activities](#).
- ⑤ **repeat** - the number of time these activities need to be scheduled. This will be visible in the [Schedule Activities](#). By default 1.
- ⑥ **yield** - the number of products *one* repeat of activities yields.
- ⑦ **user attributes** - attributes that will be assigned to the activity instance.
- ⑧ the **steadyFlow** activity is functionally equal to **steadyFlowExpanded**.



user attributes, activity phase, repeat step (and also activity trace points) are all stored in a generated Gantt Chart. See also [Schedule Activities](#)

2.5.2. Specifying the number of iterations

The number of iterations is used to calculate the throughput per resource, as explained in section [Calculating the Throughput per Resource](#). In the activity dispatching specification example above the Robot resource performs activity *MoveRobot* twice, once for the "first product" and once for the "second product". In this case the number of iterations can be used for the calculation on how many products per hour can be processed by the *Robot* resource.



The number of iterations does not influence the [Schedule Activities](#) or [Conformance Check](#).



throughput is now deprecated and can be replaced with **repeat** and **yield** as described in [Extended/Advanced Activity Dispatching Specification](#).

2.6. Create Supervisory Controller Specification

A specification with an individual activity sequence can be described in a "dispatching" specification, see [Create Activity Dispatching Specification](#). A CIF specification is used to concisely describe a supervisory controller that defines all allowed activity sequences that establish a correct product flow, while taking into account product routing flexibility.

In a CIF specification, a network of automata is used as formalism. The outgoing edges in an automaton refer to the allowed activities in the given automaton location. Multiple edges represent a scheduling choice where multiple activities are allowed. Automata are synchronized on the activity names with multi-party synchronization. This means that an activity can only be executed

in the current system state, if it is enabled in the current state of each of the synchronizing automata.

A small example of a supervisory controller is shown below.

```
controllable MoveRobot; ①

supervisor Tutorial: ②

  location l0:
    initial; ③
    edge MoveRobot goto l1; ④

  location l1:
    edge MoveRobot goto l0;

end
```

① **controllable** - specifies all activities that are used in the specification, separated by commas.

② **supervisor** - the name of the supervisory controller

③ **initial** - indicates the starting location of the supervisory controller

④ **edge** - indicates that activity *MoveRobot* can be executed from location *l0*, leading to location *l1*.

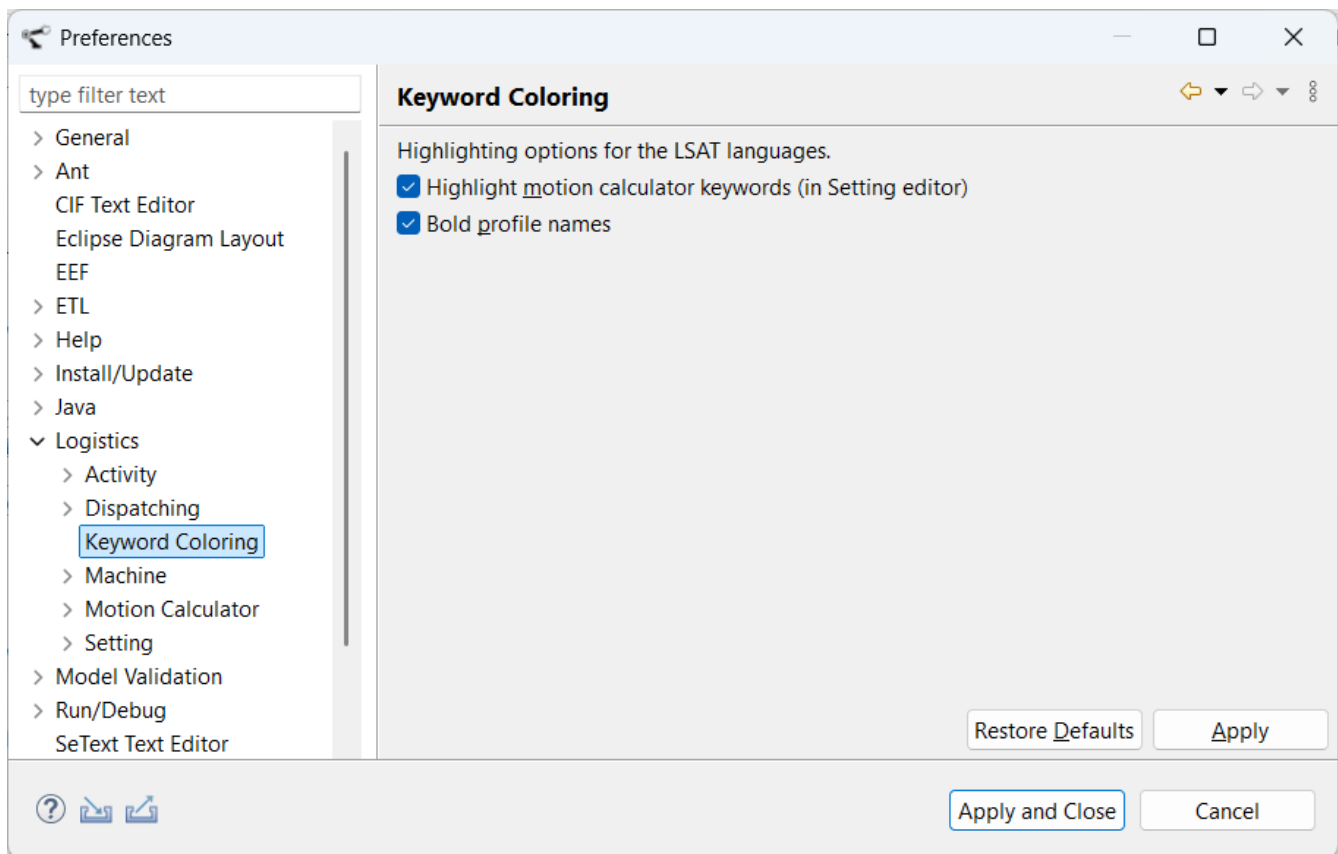
A supervisory controller can also be generated from a set of requirement and plant automata. The plant automata specify all allowed activity orderings, and the requirements put constraints on the ordering of activities to establish the correct product flow. Using supervisory controller synthesis, a supervisory controller is generated that enforces the requirements and ensures that the system does not deadlock. The documentation and tutorials for CIF, including how to use supervisory controller synthesis, can be found at <https://www.eclipse.org/escet>.

2.7. Preferences

2.7.1. Editors Configuration

Semantic Highlighting

The LSAT text editors (Activity, Machine, and Setting editors) support enhanced keyword highlighting to improve code readability and visual navigation.



The **Keyword Coloring** preferences page (accessible via **Window** › **Preferences** › **Logistics** › **Keyword Coloring**) provides two highlightable elements:

1. **Highlight motion calculator keywords (in Setting editor)** - When enabled, this option highlights motion argument names (for instance **V**, **A**, **J**, **S**) used in motion profile settings within the Setting editor. This makes it easier to distinguish between different motion parameters and their profiles at a glance.
2. **Bold profile names** - When enabled, this option highlights profile names and references across the Activity, Machine, and Setting editors with a distinctive bold italic style. This helps quickly identify which profiles are being used throughout your specifications. Examples:
 - In Machine files: profile declarations like for instance **normal** and **fast**
 - In Settings files: profile assignments in motion settings
 - In Activity files: profile references in move actions

All options are disabled by default. Enabling/Disabling any option will immediately update the highlighting in all open editors, allowing you to customize the visual presentation according to your preferences.

Chapter 3. Parameterized Activities Details

Parameterized activities allow defining reusable templates with configurable parameters for resources, positions, actions, and more. This improves flexibility, maintainability, and readability.

3.1. Defining Parameterized Activities

```
activity AnActivity(robot1:Robot1, position:SymbolicPosition, action:Action){
  prerequisites { robot1.M at position }
  actions {
    C: claim robot1
    R: release robot1
    A1: action
    ...
  }
  ...
}
```

3.2. Parameter Types

Type	Description
<Resource>	Specific resource type (e.g., Robot1)
Resource	Any resource
SymbolicPosition	Position in a Peripheral
Distance	Distance in a Peripheral
Action	Peripheral Action
Profile	Speed Profile
Event	Event to Raise or Require
<Product>	Specific product type

3.3. Using Parameterized Activities

```
activities {
  AnActivity[robot1=a, position=Above_Right, action=Robot1.a.C.clamp]
}
```

3.4. Parameter Assignment Styles

Style	Example
Named	AnActivity[robot1=a, position=Above_Right]

Style	Example
Positional	<code>AnActivity[a, Robot2, Above_Right]</code>
Full Path	<code>AnActivity[action=Robot2.C.unclamp, ...]</code>
Simplified	<code>AnActivity[action=unclamp, ...]</code> (if unique)
Item Name Only	<code>AnActivity[robot1=b, ...]</code>



Once you use a named parameter, all subsequent parameters must also be named.

3.5. Best Practices

- Match positional order to declaration.
- Prefer named parameters for clarity.
- Use descriptive names.

3.6. Examples

```
AnActivity[robot1=a, position=Above_Right, action=Robot1.a.C.unclamp]
AnActivity[a, Above_Right, Robot1.a.C.unclamp] // positional
```

3.7. Benefits

- **Reuse:** Define once, use many times.
- **Maintainability:** Update in one place.
- **Flexibility:** Adapt without duplication.

3.8. Common Issues

- Type mismatch
- Missing parameters
- Ambiguous references
- Incorrect positional order
- Mixed styles after named parameter

3.9. Validation Rules

Rule	Message
Missing parameter	No item specified for '<paramName>:<expectedType>'
Extra parameter	Unnecessary parameter specified. Please remove it

Rule	Message
Type mismatch	expecting '<ExpectedType>' but got '<ActualType>'
Ordering	Machine 'Resource' parameters must precede 'Action' parameters
Named vs positional	Scoping errors
Scoping	Action must belong to claimed resource
Unique names	Duplicate parameter name <name>
Unused parameter	<name> not used
Mixing params & incomplete resources	Activity <name> has incomplete resource items AND parameters
Resource without items	Resource <name> has no resource items
Path completeness	Resource → Robot1.a, Action → Robot1.a.C.clamp
Short vs full path	Short allowed if unique; full always valid
Passive claim	Claim <C> can be made passive. There are no actions for resource <resourceName>

3.10. Quick Syntax Examples

For activity:

```
activity AnActivity(resource:Resource, action:Action){
    ...
}
```

□ Correct:

```
AnActivity[resource=Robot1.a, action=Robot1.a.C.clamp]
AnActivity[Robot2, Robot2.C.clamp] // positional allowed if none named before
```

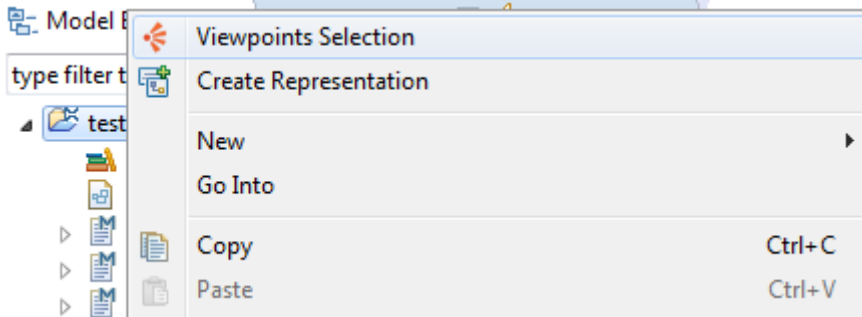
□ Incorrect:

```
AnActivity[action=Robot1.a.C.clamp, resource=Robot2] // Resource after Action
AnActivity[resource=Robot1.a] // Missing action
AnActivity[resource=Robot2, action=Robot2.C] // action ends at Peripheral
AnActivity[resource=Robot2, action=Robot2.C.clamp, extra=a] // Extra param
```

Chapter 4. Graphical Viewers

4.1. Add Viewpoints Selection

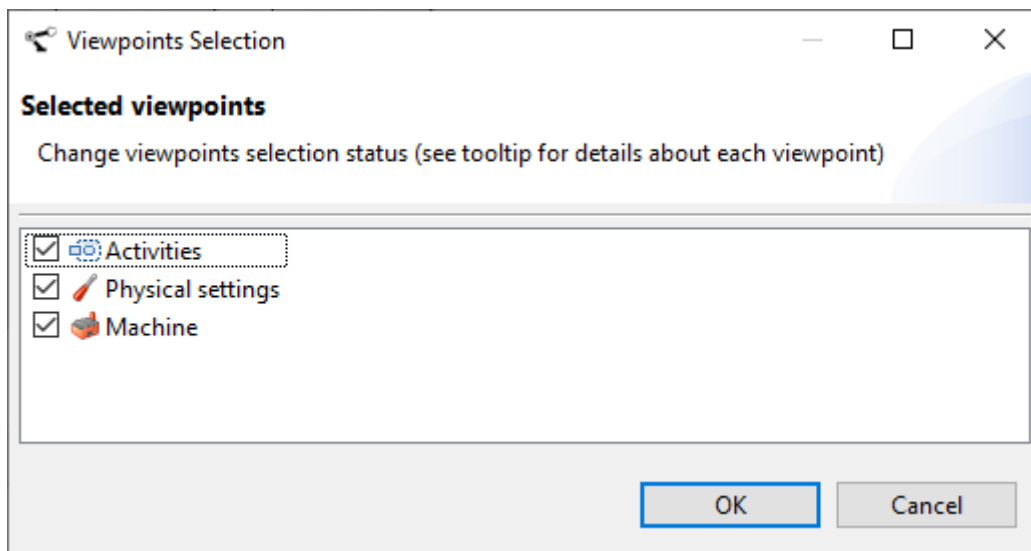
Right click on the project folder, select Viewpoints Selection.



The following viewpoints are available:

- Machine** shows the resources with their peripherals and illustrates symbolic position graphs for movement-based peripherals.
- Activities** shows activity diagrams based on activity specifications.
- Physical settings** shows settings for motion profiles and physical locations.

Check the relevant viewpoints and click OK.

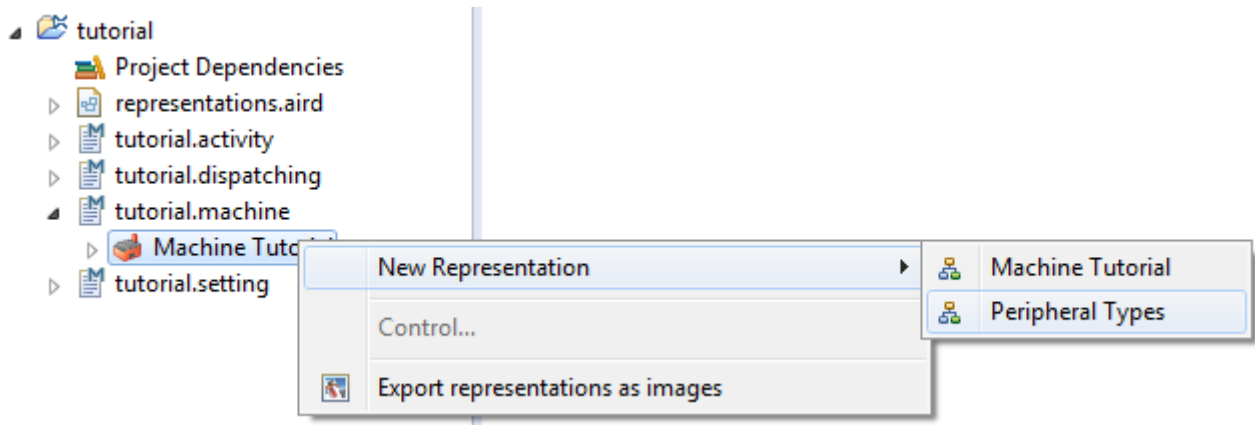


4.2. Generate Graphs

4.2.1. Peripheral types in machine

Target element: a machine

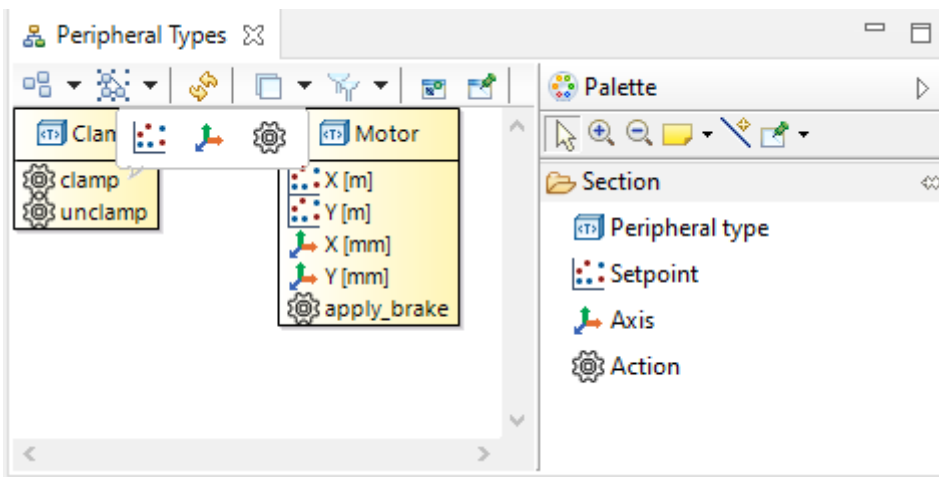
Right click on a target element, select New Representation and choose the corresponding Peripheral Types for the target element.



The peripheral types graph for the target element includes all action and movement based peripheral types for the target element.



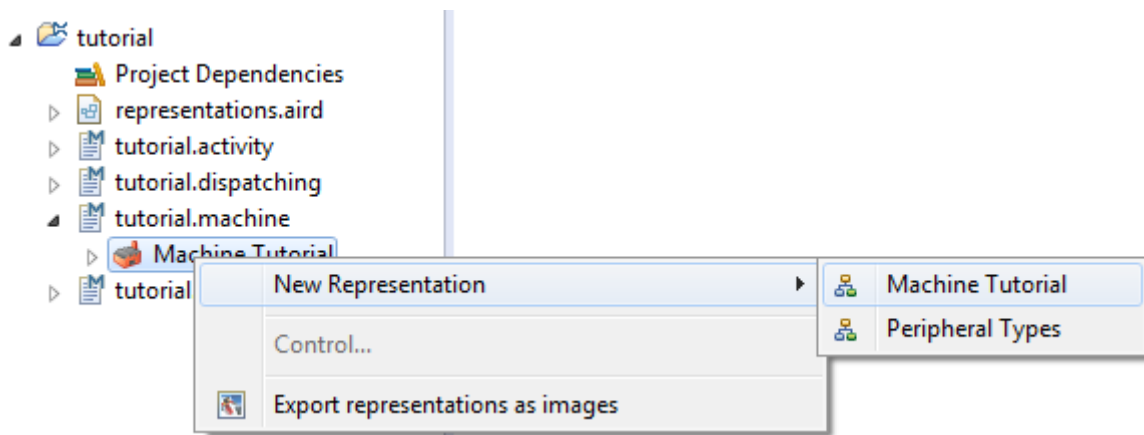
This viewer can also be used to edit the peripheral types by using the palette.



4.2.2. Resources and peripherals in machine

Target element: a machine

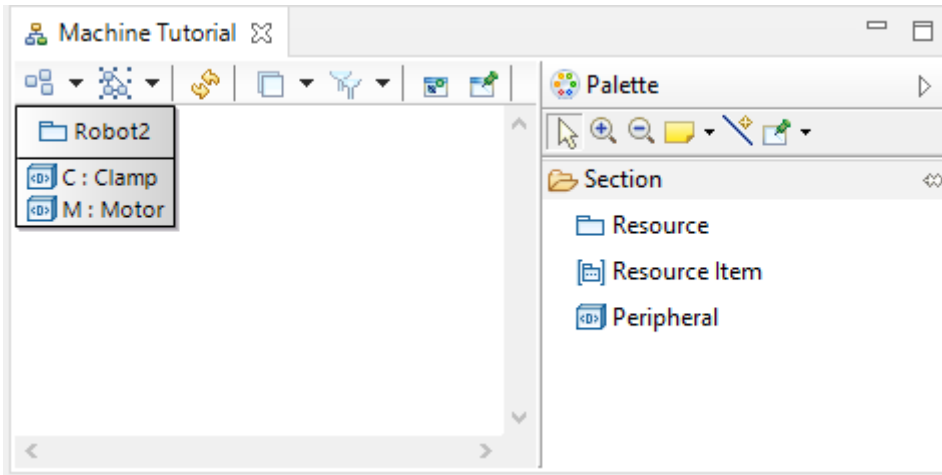
Right click on a target element, select New Representation and choose the corresponding Machine for the target element.



The graph for the target element includes all resources with their action-based and movement-based peripherals for the target element.



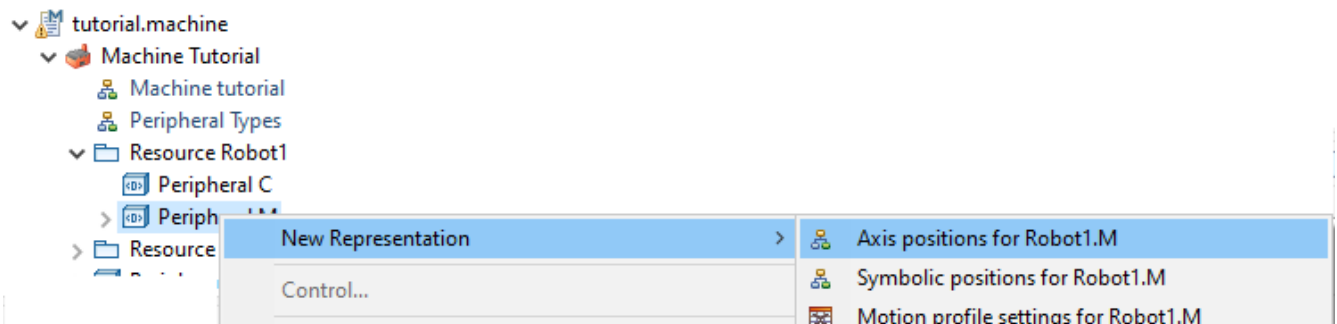
This viewer can also be used to edit the resources and their peripherals by using the palette.



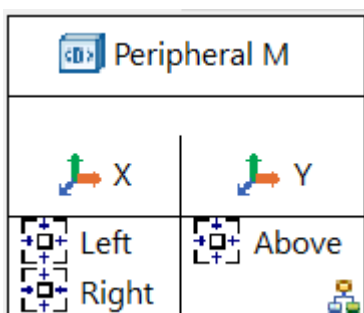
4.2.3. Axis position graph for peripheral

Target element: a movable peripheral defined in a resource

Right click on a target element, select New Representation and choose the corresponding Axis positions for the target element.



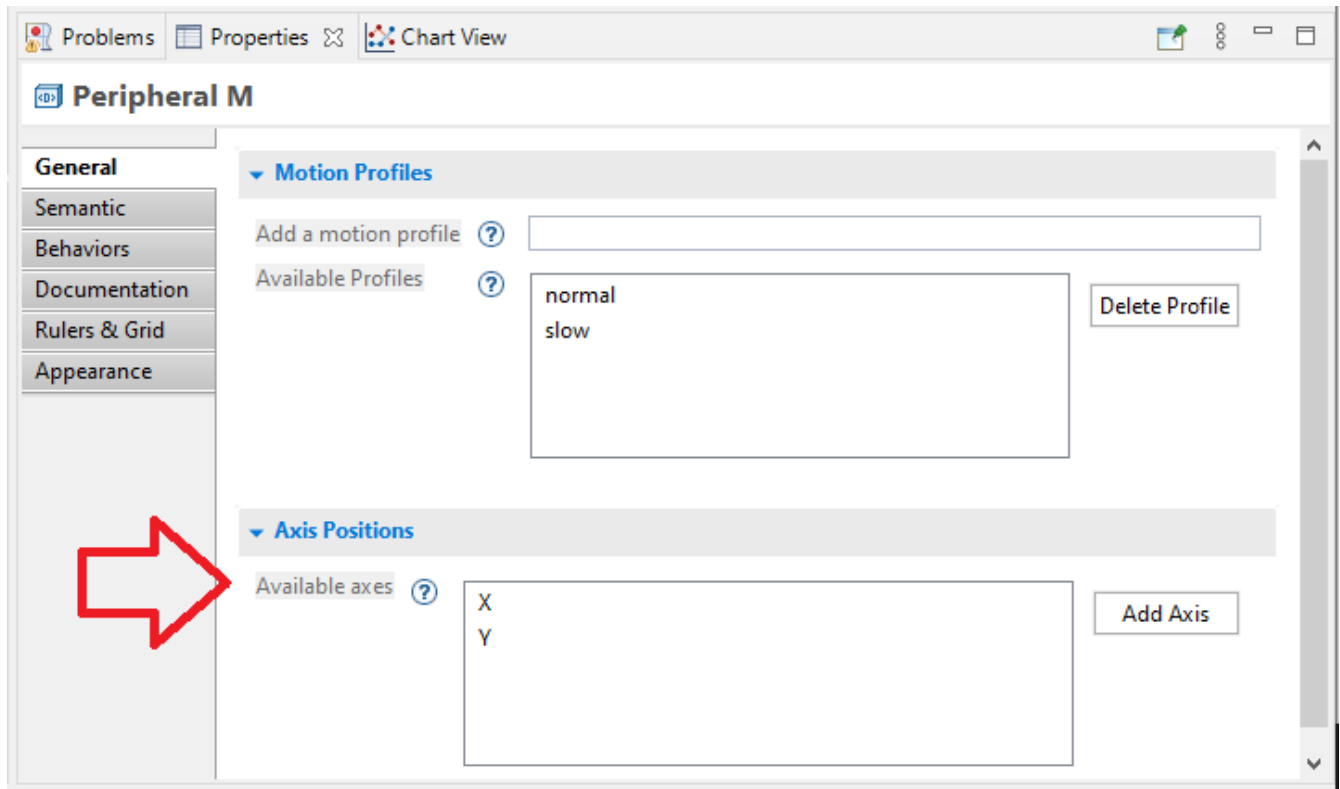
The axis position graph for the target element includes specific information about axes and corresponding positions.



The axes in a axis position graph are shown as compartments, and the positions associated to an axis are shown under the axis. For example, in the figure above, the position *Above* is associated to the axis Y.

Adding/removing axes

To add a new axis to a peripheral, select the empty canvas and select the desired axis from the *Available axes* list in the properties view. Press *Add Axis* on the right side.



To remove an axis from a peripheral, select it and press Delete.



Deleting an axis will also delete all corresponding positions.

Adding/deleting positions

The positions can be associated to an axis by selecting the *Position* tool from the *Axis Positions Editor Tools* palette on the right side, and selecting the axis compartment.

By default, name of the newly created position is *new*. The name of a position can be changed by selecting it, typing the name directly and pressing Enter. Alternatively, the name can be changed using the properties editor, as shown in the figure below.

Semantic	Property		Value
	Position Above		
	Name		Above

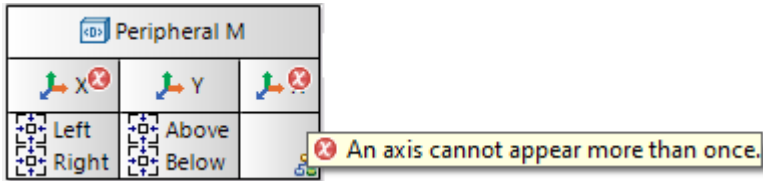
To delete a position, select it and press Delete.



If a position is specified in both axis position graph and Settings [Create Settings Specification](#), deleting that position from the axis position graph will also delete it from the Settings specifications.

Validation

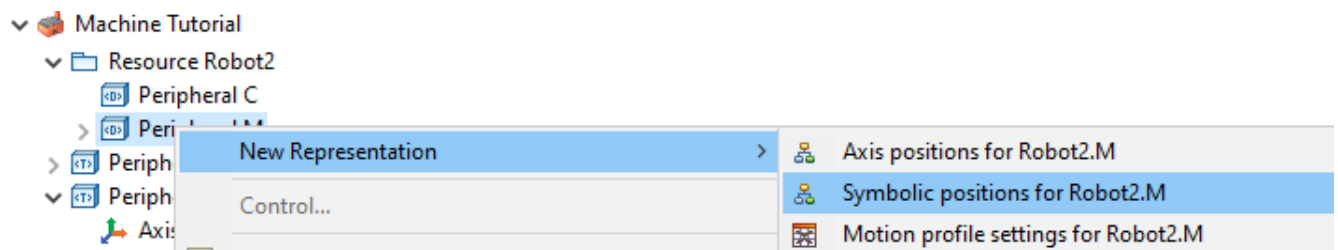
An axis must not be added more than once. To validate this property, right click on the canvas and click *Validate diagram*. If any axis is added more than once, a red error sign will appear on those axes, as shown in the following figure.



4.2.4. Symbolic position graph for peripheral

Target element: a movable peripheral defined in a resource

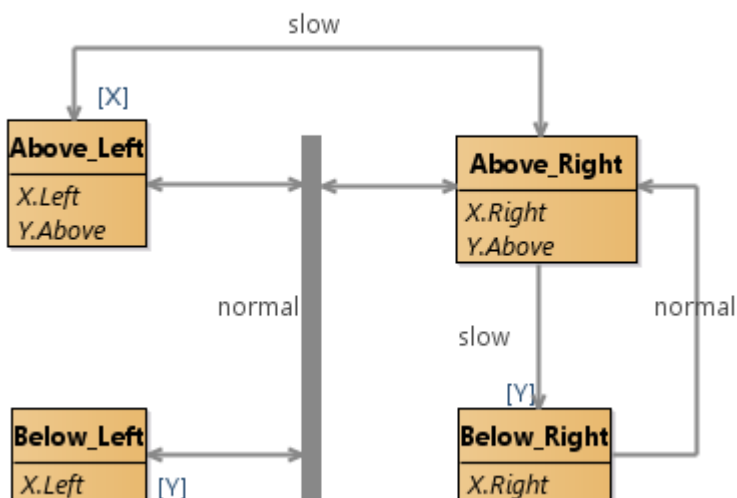
Right click on a target element, select New Representation and choose the corresponding Symbolic positions for the target element.



The symbolic position graph for the target element includes specific information about profiles and settling on the paths.



Layout of existing symbolic position graphs will be reset to default layout. As manual layout of these diagrams may require extra time and labor, it may not be wise to upgrade to the latest release of Eclipse LSAT™ for existing symbolic position specifications.



The symbolic positions are shown as compartments. The axis positions corresponding to a symbolic

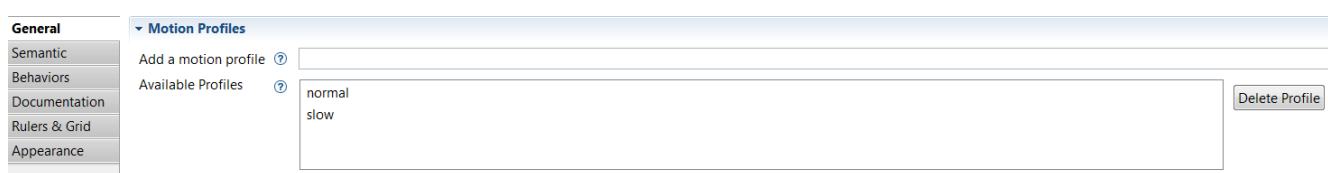
position are shown inside the respective compartment.

The full mesh paths are shown as gray vertical rectangles. The single-headed arrows between symbolic positions represent unidirectional paths, whereas double-headed arrows represent bidirectional paths. If a symbolic position is a part of a full mesh path, then there is also an arrow between the symbolic position and the full mesh path.

The labels in the center of paths represent motion profiles. Whereas, the labels at the end of paths represent settling.

Adding/deleting motion profiles

To add a new motion profile to a peripheral, click on the empty canvas. As a result, properties editor will appear at the bottom. Enter the name of the motion profile without a white space in the field *Add a motion profile*, as shown in the following figure, and hit Enter.



The screenshot shows the 'Motion Profiles' panel. On the left is a sidebar with tabs: General, Semantic, Behaviors, Documentation, Rulers & Grid, and Appearance. The 'Motion Profiles' tab is active. It contains two input fields: 'Add a motion profile' and 'Available Profiles'. The 'Available Profiles' field contains the text 'normal' and 'slow'. To the right of the 'Available Profiles' field is a 'Delete Profile' button.

The available profiles are visible in the *Available Profiles* field. To delete a profile, select it and press *Delete Profile* on the right side.

Adding/deleting symbolic positions

To create a new symbolic position, select the respective tool from the *Symbolic Positions Editor Tools* palette on the right side, and then click on the canvas. The name of the symbolic position can be typed directly while selecting the newly created symbolic position. Alternatively, the name can also be typed in the properties editor at the bottom, as shown in the following figure.



The screenshot shows the 'Properties' editor for a 'Symbolic Position Below_Left'. The editor has a sidebar with tabs: Semantic, Style, and Appearance. The 'Semantic' tab is active. It displays a table with the following data:

Property	Value
Symbolic Position Below_Left	
Name	Below_Left
Source References	
Target References	Path Target Reference Below_Left

To delete a symbolic position, select it and press Delete.

Adding/deleting axis positions

The axis positions can be added to a symbolic position compartment by selecting the respective tool and selecting the symbolic position compartment. The name of the axis position can be typed directly in the format *Axis.Position* (same as textual representation). Alternatively, the name can also be typed in the properties editor at the bottom, as shown in the following figure. Here, *Key* represents an axis and *Value* represents a position. To delete a axis position, select it and press Delete.

	Property	Value
Semantic	machine.impl.AxisImpl@7af33753 (name: X, unit: mm) -> machine.impl.Position	
Style	Key	Axis X
Appearance	Value	Position Right



The typed axis position name cannot be entered if either of the typed axis or position does not exist.

Adding/deleting paths

Uni/bidirectional paths between two symbolic positions can be added by selecting the respective tool and clicking the symbolic positions. To add a symbolic position to a full mesh path, select the *Full Mesh Target* tool, and then click the full mesh path and the symbolic position. To delete a path, select it and press Delete.

A settling to a unidirectional path can be added in the properties editor by selecting the path, as shown in the figure below.

▼ Settling

Assign a settling ?

Current settlings ?

Delete Settling

In the *Assign a settling* field, a settling can be selected for a path. The settlings currently assigned to a path are visible in the *Current settlings* field. To remove a settling from a path, first select the path and then select the settling in the *Current settlings* field and press *Delete Settling* on the right side.

In bidirectional paths, we can add two settlings, i.e., one settling for each connected symbolic position. A settling to a bidirectional path can be added in the properties editor by selecting the path, and then assigning a settling at each connected symbolic position, as shown in the figure below.

▼ Settlings

Assign a settling at the location: Above_Left ?

Current settlings at the location: Above_Left ?

Delete Settling

Assign a settling at the location: Above_Right ?

Current settlings at the location: Above_Right ?

Delete Settling

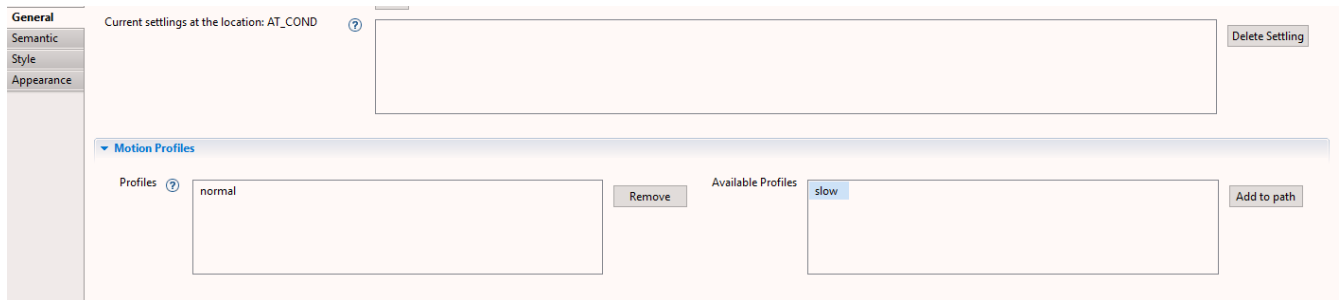
The settlings currently assigned to a path at a symbolic position are visible in the *Current settlings* field. To remove a settling from a path, first select the path and then select the settling in the *Current settlings* field and press *Delete Settling* on the right side.

Assigning existing motion profiles to a path

To add a motion profile to a path, choose one of the options below:

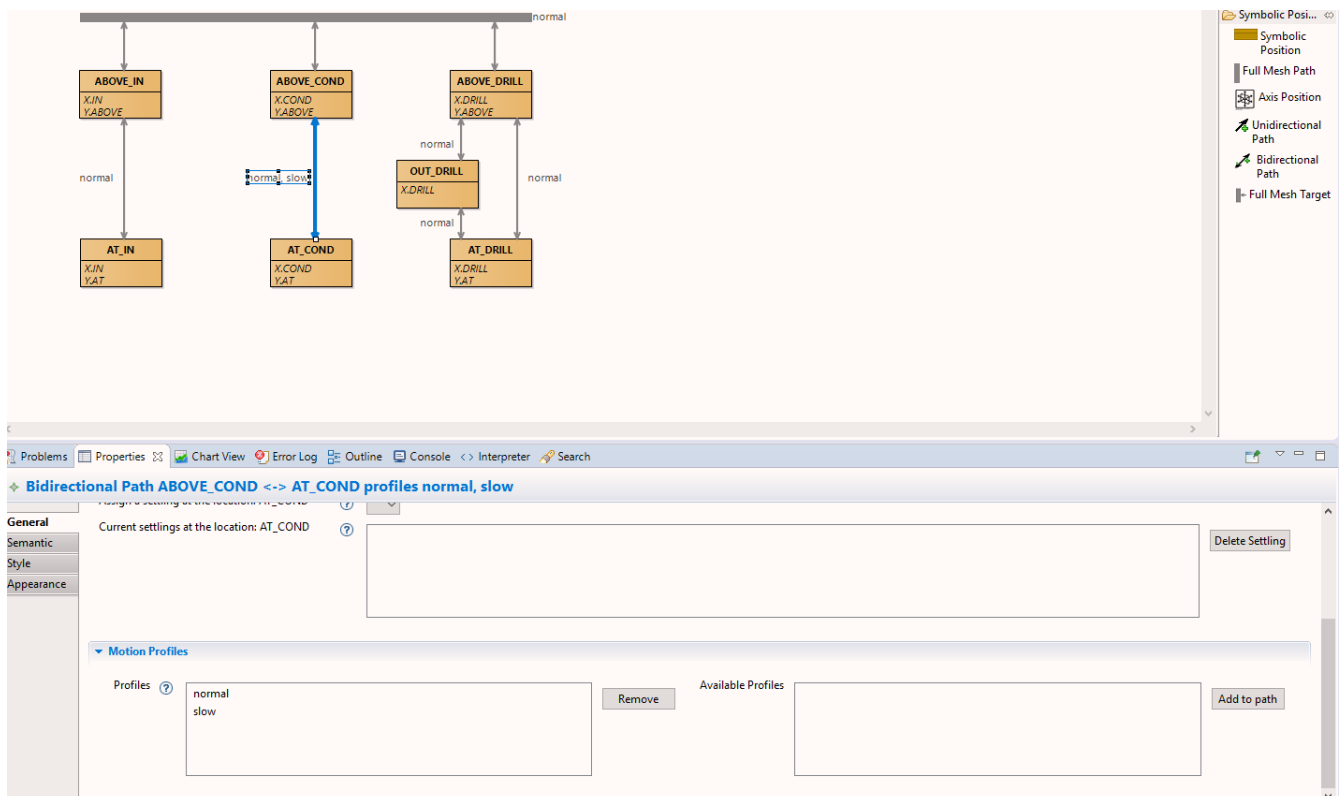
Option I

- Select the path by left clicking on it.
- Select the *General* tab.
- Scroll down to the *Motion Profiles* section.
- Select your the profile you want to add from the *Available profiles* list.
- Press Add to path.



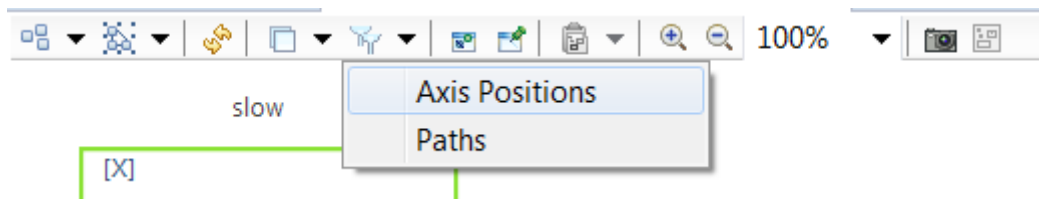
Option II

- Select the motion profile from the diagram by left clicking on it.
- Press *F2* to enable direct editing.
- Add a comma after the existing profile and write the name of the profile you want to add, e.g *normal, slow* .Please note *slow* should have been initialized earlier.
- Save your diagram. The changes are then saved to the model.



Hiding axis positions/paths

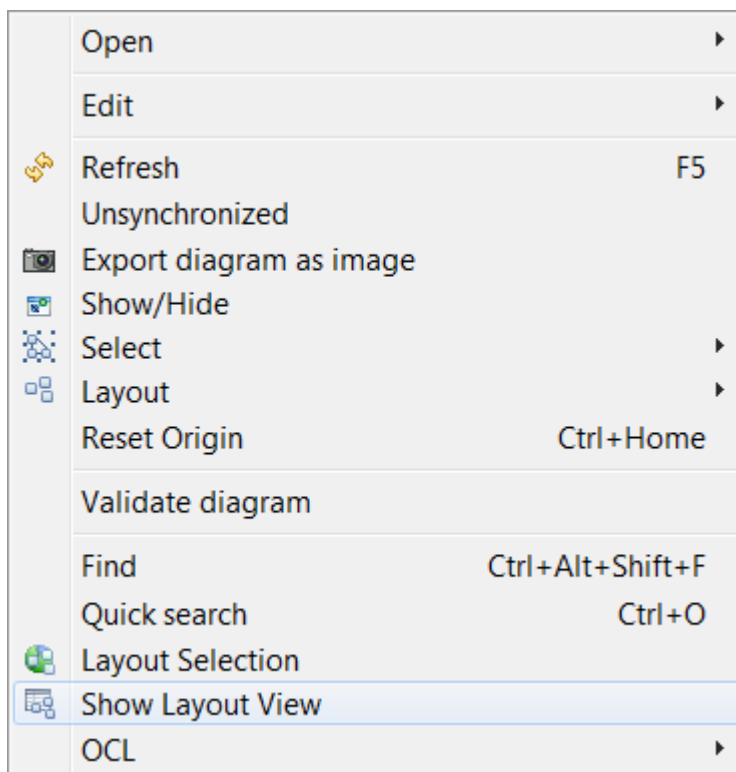
In order to visualize a diagram clearly, it is possible to hide axis positions or paths or both from a diagram. To hide axis positions, click on the canvas, and then click on the *Filter* icon on the top tool bar, and click *Axis Positions*, as shown in the figure below.



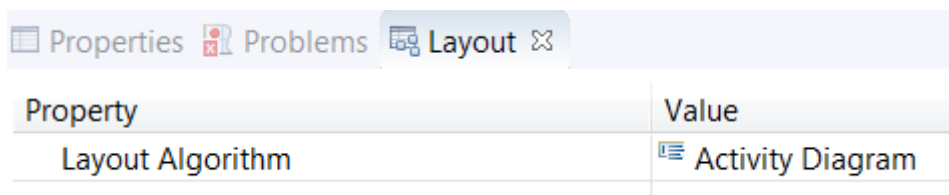
Paths from a diagram can also be hidden in the same way.

Layout

A symbolic position graph can be layout by right clicking on the canvas and selecting Show Layout View.



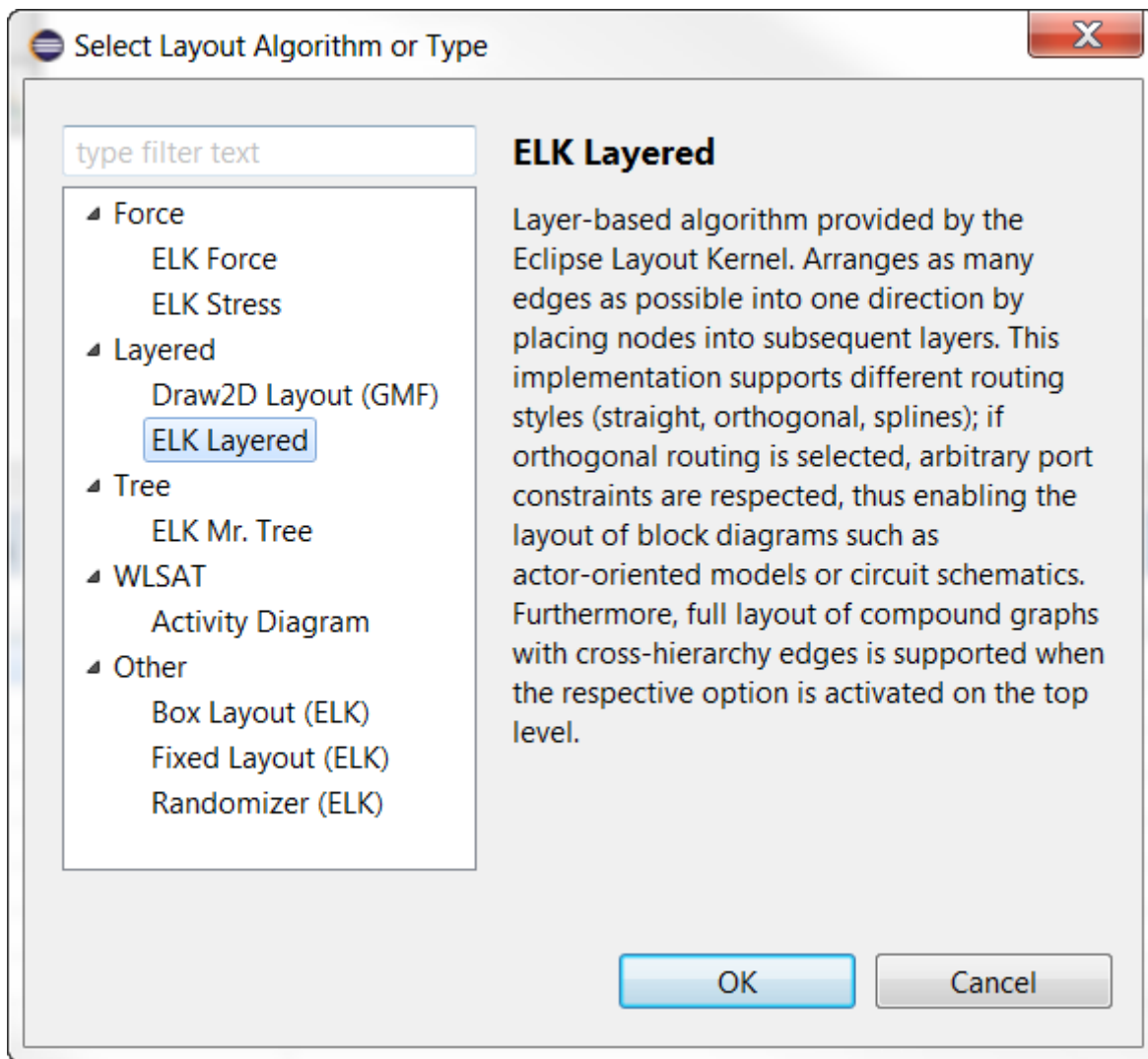
As a result, a new tab named *Layout* will appear at the bottom. Click on the empty canvas, and then in the *Layout* tab, a new Property *Layout Algorithm* with a Value *Activity Diagram* will appear.



Select *Activity Diagram*, and double click it or use the button on the right side.

A pop-up will appear showing all pre-defined layout algorithms. In our experience, the *ELK Layered* algorithm is most suitable for symbolic position graphs. However, please feel free to explore other

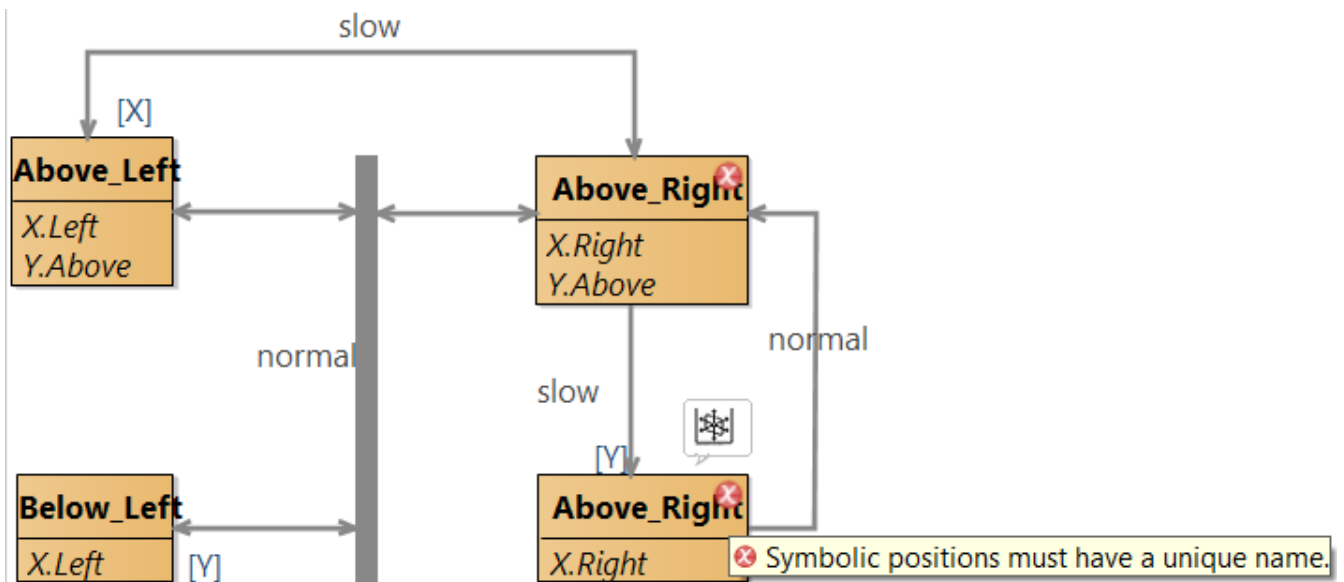
algorithms also.



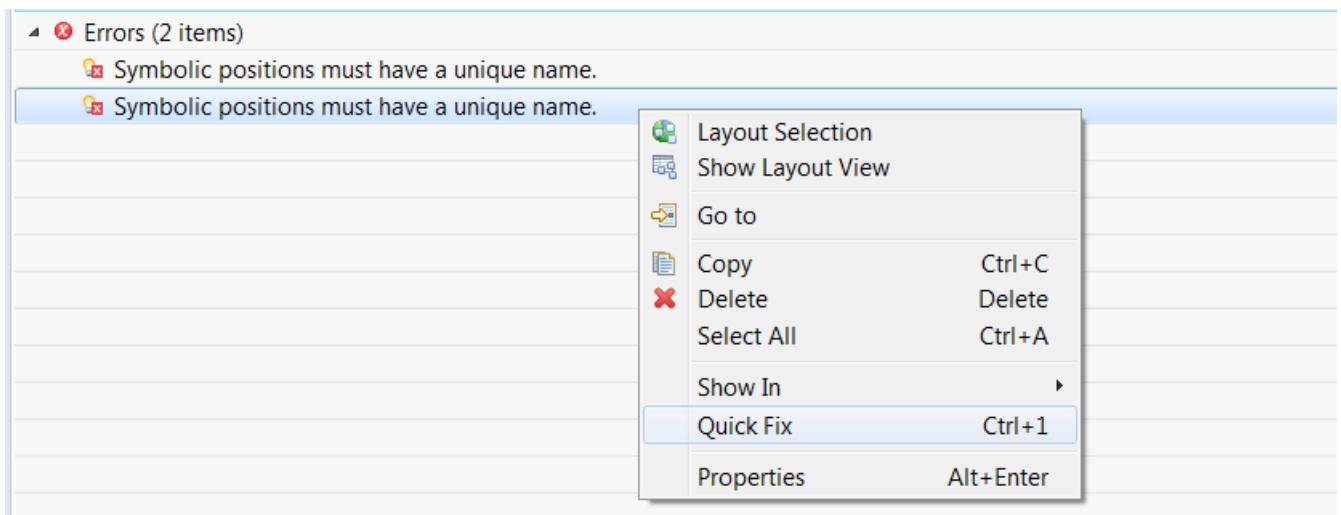
Select an algorithm and press OK. Afterwards, apply the automatic layout by right clicking on the canvas and selecting Layout Selection from the context menu, or use the button from the toolbar .

Validation

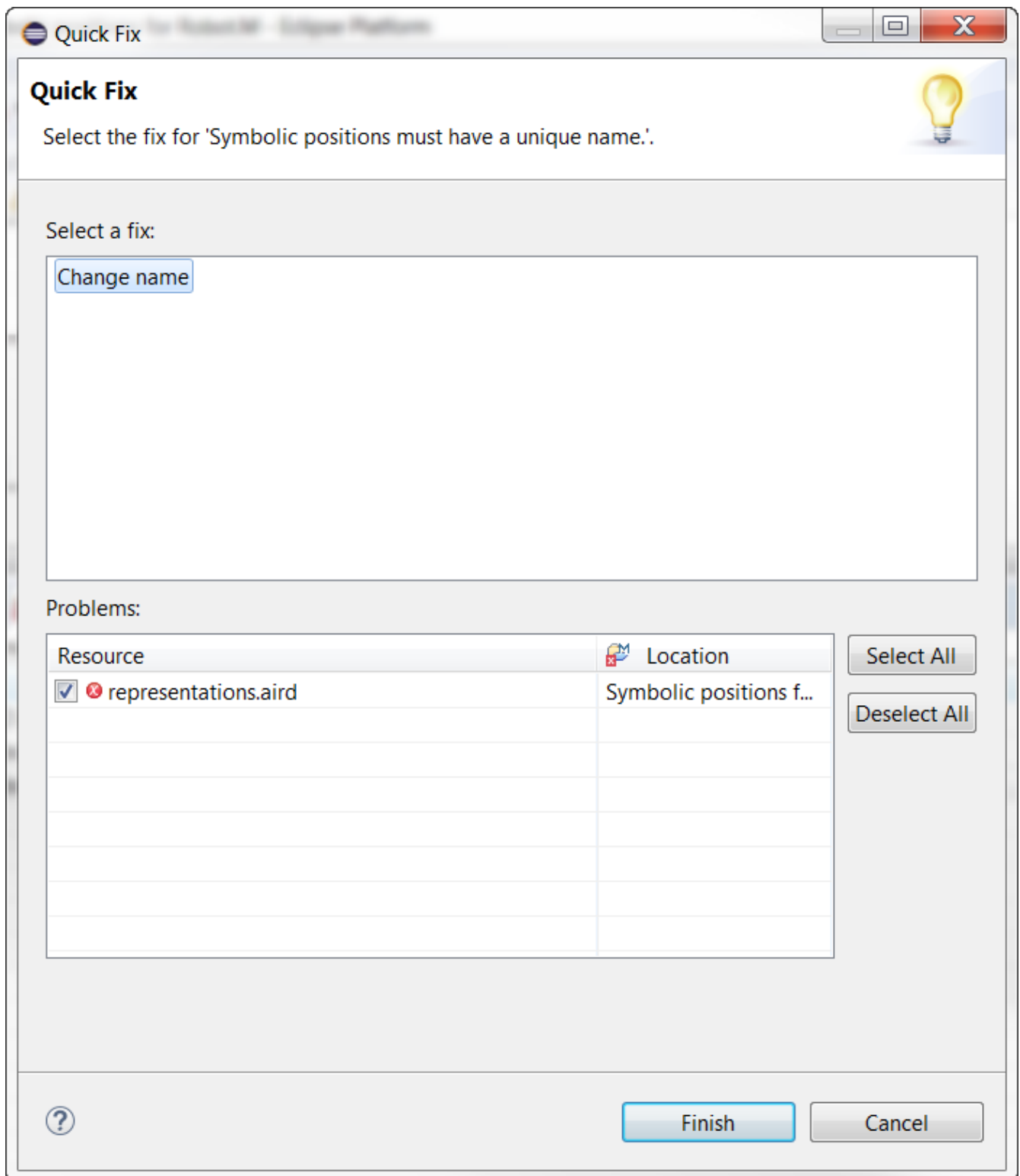
Each symbolic position must have a unique name. To validate this property, right click on the canvas and click *Validate diagram*. If any symbolic positions do not have unique names, a red error sign will appear on those symbolic positions, as shown in the following figure.



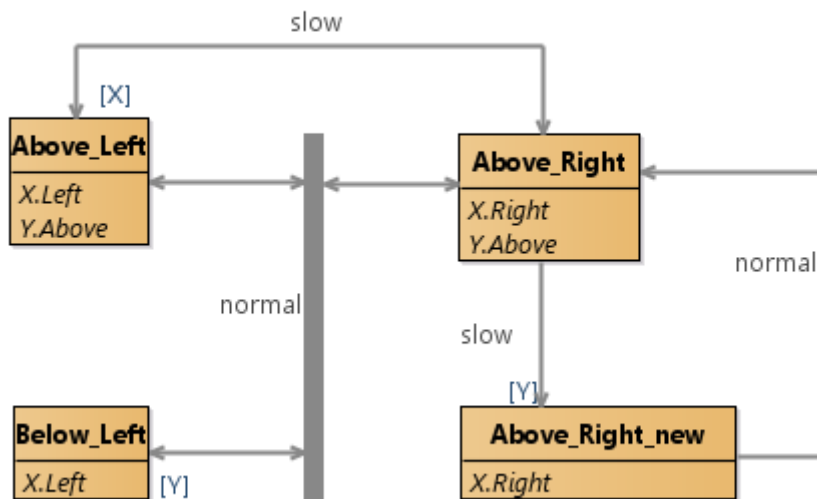
To quickly fix this error, a quick fix is available. Go to the *Problems* tab on bottom and right click on the error message and select *Quick Fix*, as shown in the figure below.



Then select the *Change name* and press Finish.



As a result *_new* will be added at the end of the name of the symbolic position.



4.2.5. Motion profile settings table for peripheral

Target element: a movable peripheral defined in a resource.

Right click on a target element, select New Representation and choose the corresponding motion profile for the target element.



If the representation is not listed in the context menu, most likely the *physical settings* viewpoint needs to be activated for the project, see [Add Viewpoints Selection](#).

Motion profile settings data is from the first setting file in the project directory.

Column specifies profile names and row represents one axis. Each row contains sub rows to specify VAJs for an axis.

Motion profile settings for Robot.M			
	Normal	Slow	
X			
V [m/s]	1	0.5	
A [m/s^2]	1	0.5	
J [m/s^3]		0.5	
Y			
V [m/s]	1.0	0.5	
A [m/s^2]	1.0	0.5	
J [m/s^3]		0.5	

4.2.6. Physical location settings table for peripheral

Target element: a movable peripheral defined in a resource.

Right click on a target element, select New Representation and choose the corresponding physical location settings for the target element.



If the representation is not listed in the context menu, most likely the *physical settings* viewpoint needs to be activated for the project, see [Add Viewpoints Selection](#).

The same to motion profile settings table, the physical location settings data is also from the first setting file in the project directory.

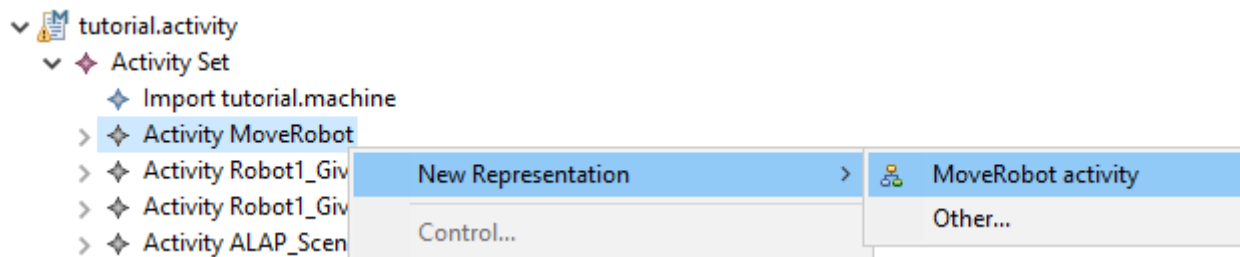
In the table, each column represents one symbolic position for the target element. Each row represents its axes and the physical location values (min, default, and max) are shown on the sub rows.

Physical location settings for Robot.M				
	Above_Left	Above_Right	Below_Left	Below_Right
X [mm]	Left	Right	Left	Right
min	-100		-100	
default	-150	300.0	-150	300.0
max	-200		-200	
Y [mm]	Above	Above	Below_Left	Below_Right
min				
default	150.0	150.0	-160.0	-150.0
max				

4.2.7. Activity diagram

Target element: an activity defined in an activity specification

Right click on a target element, select New Representation and choose the corresponding activity for the target element.



If the representation is not listed in the context menu, most likely the *activities* viewpoint needs to be activated for the project, see [Add Viewpoints Selection](#).

It shows an activity diagram with an action flow. Green swim-lanes represent resources and their peripherals, gray blocks represent either claiming a resource or releasing a resource and brown blocks represent actions for peripherals. If an action is specified to be scheduled As Late As Possible (ALAP), then an icon with an arrow pointing down is shown.

The generated activity diagram (probably) needs some layout to be done, but an automated default layout is applied. The initial layout is not optimal yet, therefore it is wise to apply the automatic layout once the Activity diagram is visible, using the tip below.

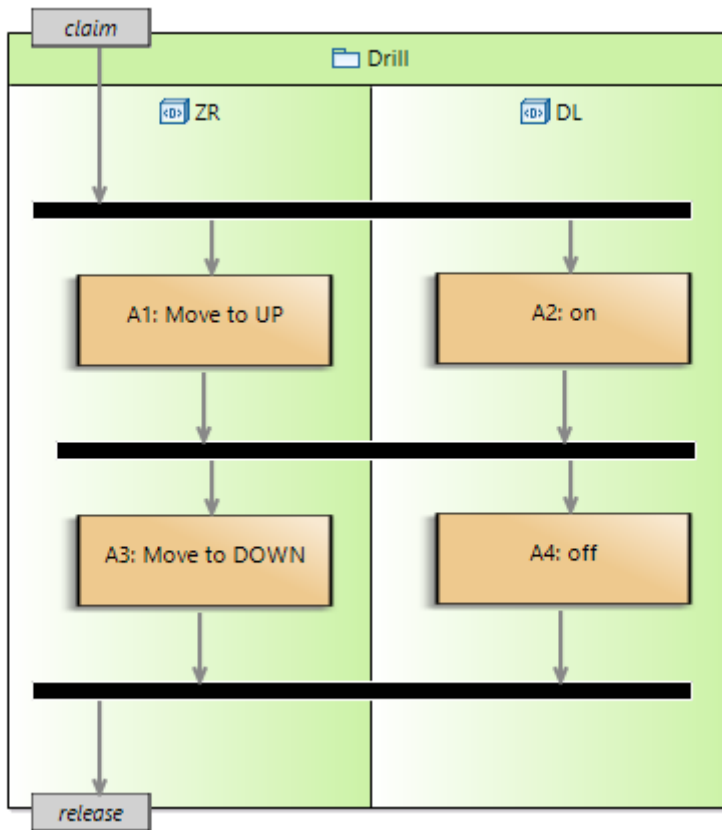


To (re)apply the automatic layout, select *Layout Selection* from the context menu

on the canvas, or use the button from the toolbar 



This viewer can also be used to edit the activity by using the palette.



Adding a new action

Action details
Provide the details for the action to add

Name: A5

☐ Action:

☒ Move: Symbolic Position UP

Move type: Point-to-point (stop at target position)

Speed Profile: Profile normal

Scheduling type: As Soon As Possible (ASAP)

Resource: Drill

? < Back Next > Finish Cancel

1. Select **Action** from the Palette at the right side of the editor.

2. Left click somewhere in the green swim-lane of a peripheral. A popup window as the one above will show up.
3. Provide a name for the action.
4. Use the radio button to select the type of action you want to create. If the action is of type *Move*, then you must also specify if it is point-to-point (stop at target position) or on-the-fly (don't stop at target position) and the speed profile to use.
5. Select the scheduling type - As soon as possible (ASAP) or As late as possible (ALAP)
6. Click Finish.
7. [New action](#) illustrates an example. In this case, the new action is A7.

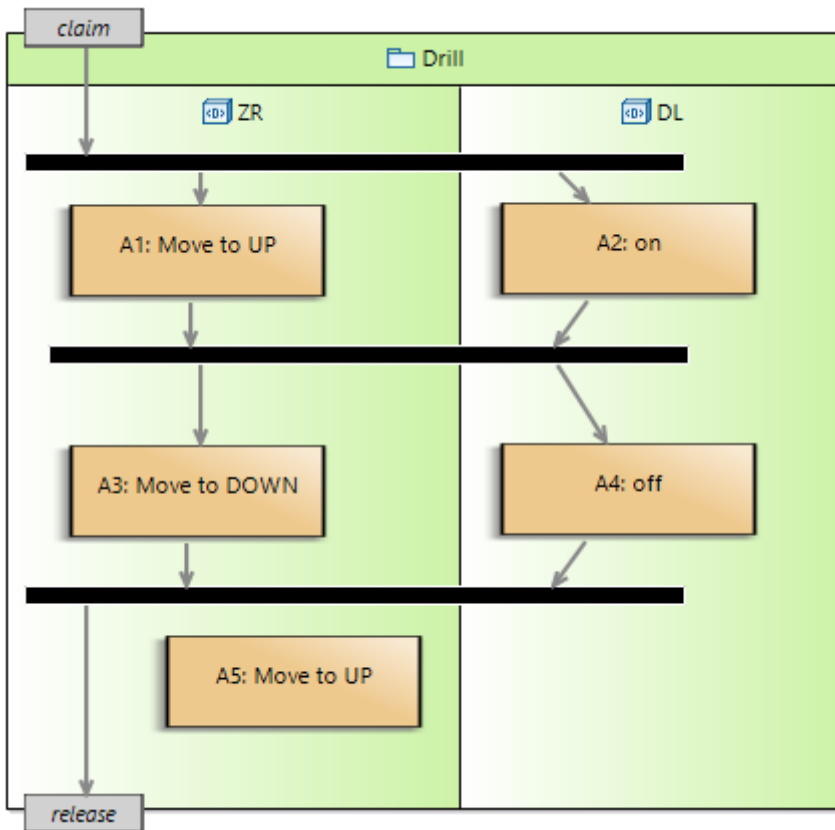


Figure 9. New action

Modifying the properties of an existing action

1. Select the action you would like to modify from the activity diagram.
2. Select the **Properties** tab in Eclipse. It is usually located below the diagram.
3. You should now have access to a menu that looks like the image below.

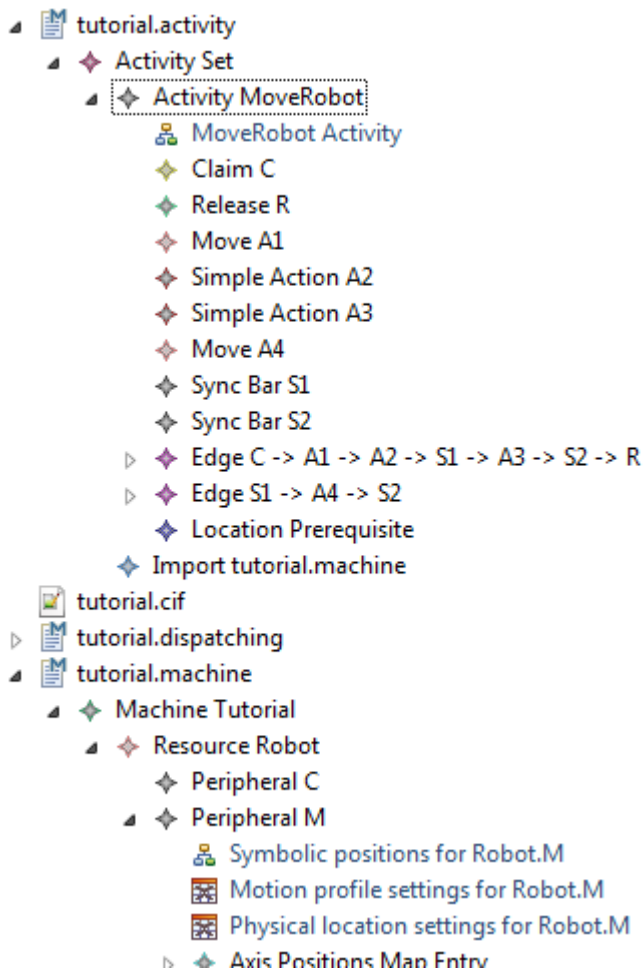
Semantic	Property	Value
Style	Continuing	☐ false
	Distance	☐
	Graph	☐ Activity Drill
	Incoming Edges	☐ Edge S2 -> A3 -> S3 -> R1
	Name	☐ A3
	Outgoing Edges	☐ Edge A3 -> S3 -> R1
	Passing	☐ false
	Peripheral	☐ Peripheral ZR
	Position Move	☐ true
	Predecessor Move	☐ Move A1
	Profile	☐ Profile normal
	Resource	☐ Resource Drill
	Appearance	Scheduling Type



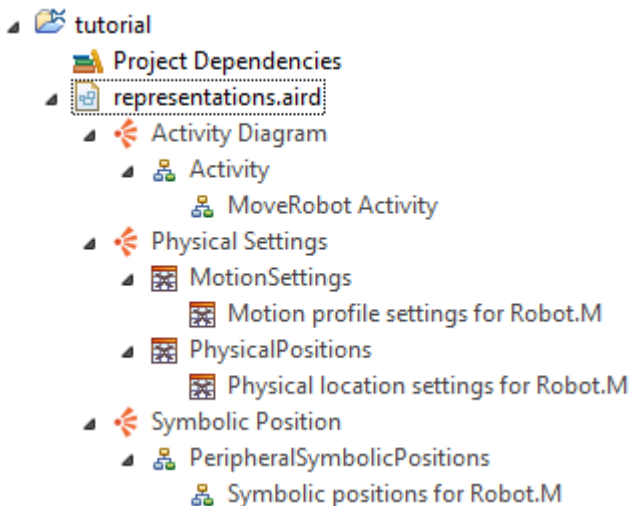
You can use the **Properties** view to change the scheduling type of an action from **ASAP** to **ALAP** or vice versa.

4.3. Location of Generated Graphs

The generated graphs are located inside its target element. By unfolding the element from the Model Explorer, the generated graphs will appear.



All the graphs can also be found under the “representations.aird” file.



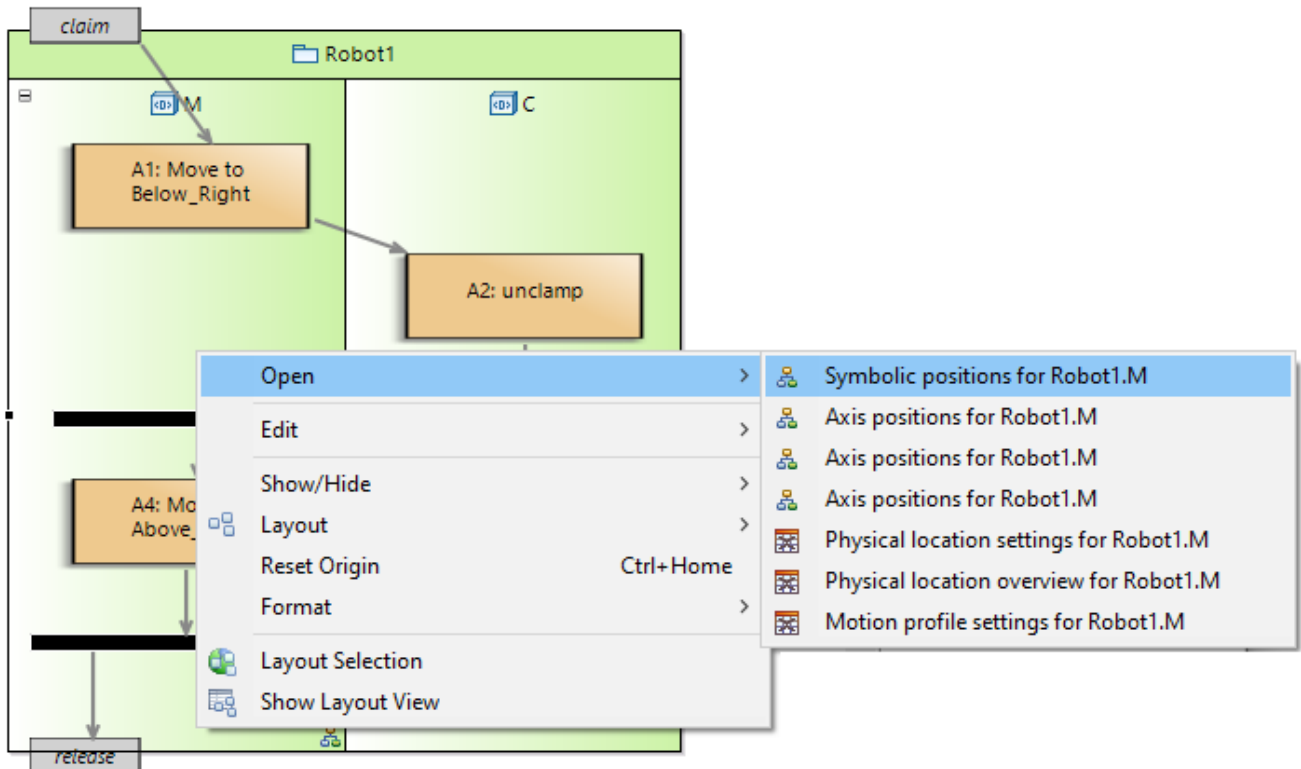
4.4. Navigation among graphs

In graphical viewer, if one semantic element can be generated more than one graph, these graphs can be navigated. Navigation only happens when the semantic element is shown as a container/block in a source graph. Since there is no container concept in table, there is no navigation starting from a table view. Besides, navigations also request the target graphs have already been generated.

4.4.1. Navigate from activity diagram

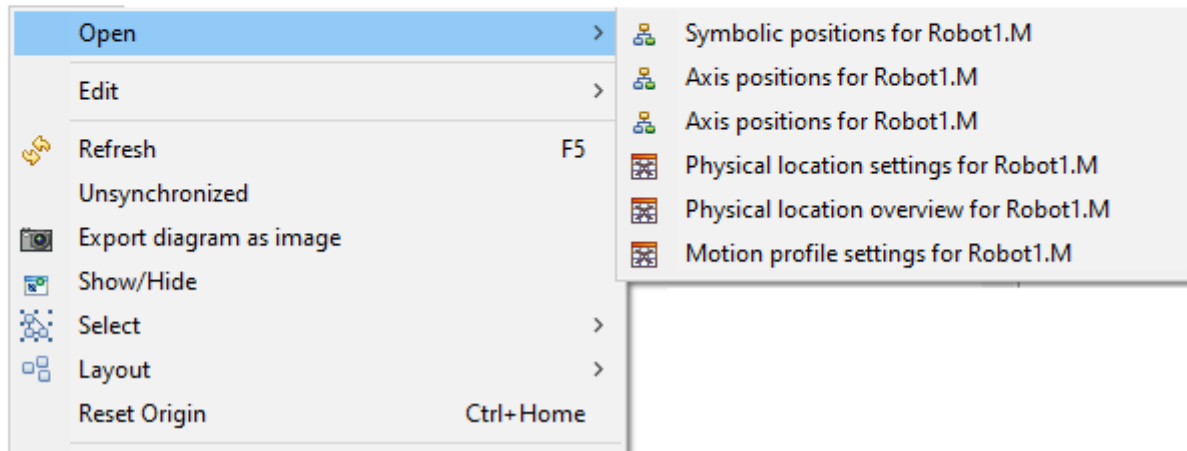
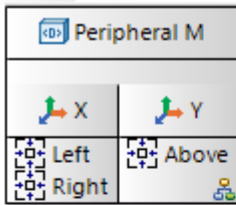
A peripheral is represented as a write block in an activity diagram. Some white blocks contain a small graph icon  at right bottom corner. It means there are other graphs can be generated for the semantic element of this block.

Right click on the write block, open with a desired graph. Movable peripherals can produce five types of graphs: axis position graph, symbolic position graph, motion profile settings table, physical location settings table and physical location overview table.



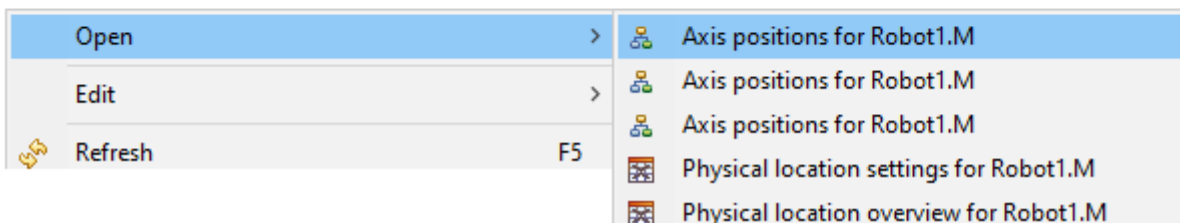
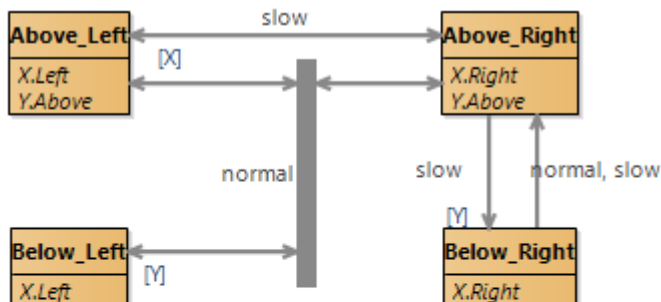
4.4.2. Navigate from axis position graph

It is the same as navigation from activity diagram. Right click the container which has a small graph icon and open with a desired graph. From a axis position graph, it is possible to navigate to symbolic position graph, motion profile settings table, physical location settings table and physical location overview table.



4.4.3. Navigate from symbolic position graph

Right click the container which has a small graph icon and open with a desired graph. From a symbolic position graph, it is possible to navigate to axis position graph, motion profile settings table, physical location settings table and physical location overview table.



Chapter 5. Schedule Activities

5.1. Scheduling & Configurations



Scheduling an activity sequence requires a fully specified system. See section [Logistics Specification](#) for more information on creating this specification.

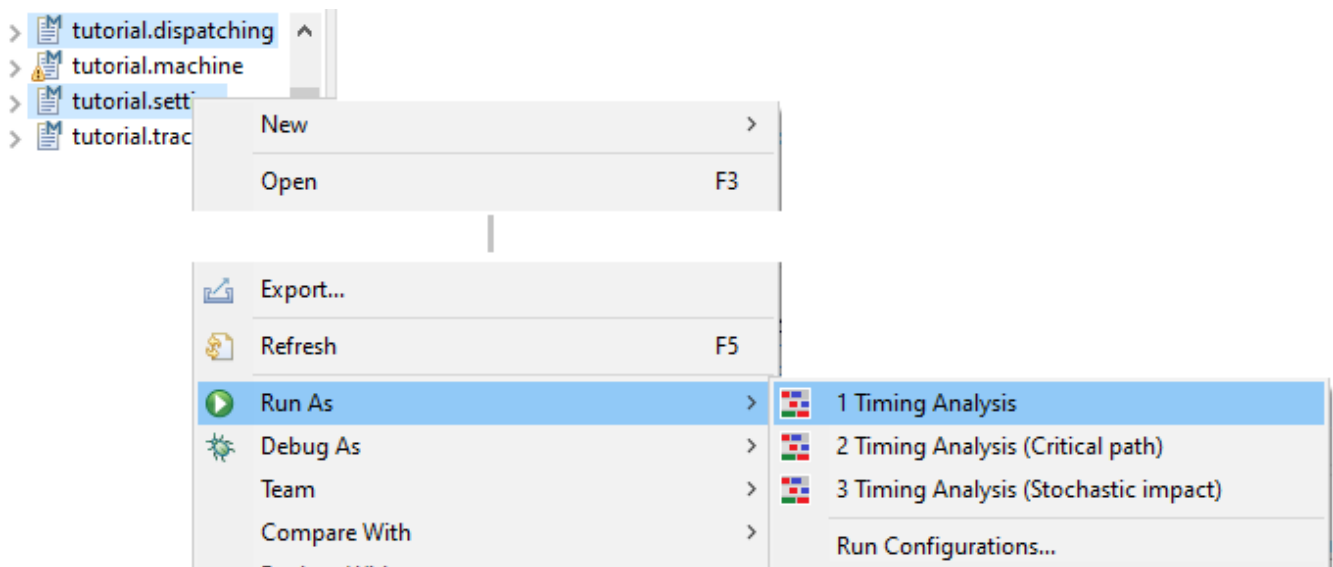
Scheduling an activity sequence generates a Gantt chart that represents the system execution in time. Within an activity sequence one or more activities can be included. These activities will be scheduled as parallel as possible adhering to the claim/release strategy. If activities cannot be scheduled in parallel the order of this section is applied. An activity typically starts with claiming a Resource, hence this activity can only start if its predecessor activity has released this Resource.

Optionally an offset can be defined for an activity sequence. The scheduler will ensure that these activities will not be started before the offset time.

The easiest way to run a Timing Analysis is to select a dispatching file *and* a compatible setting file. Then right click and select Run As/Timing Analysis.

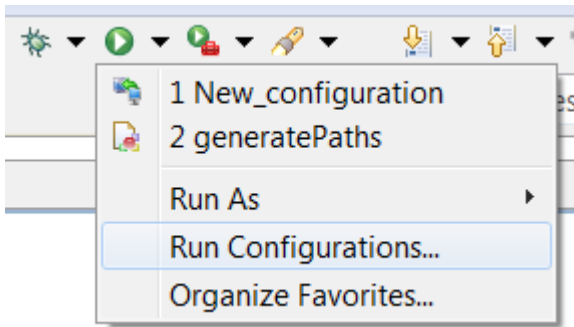


Use <ctrl> <click> to select more than one item

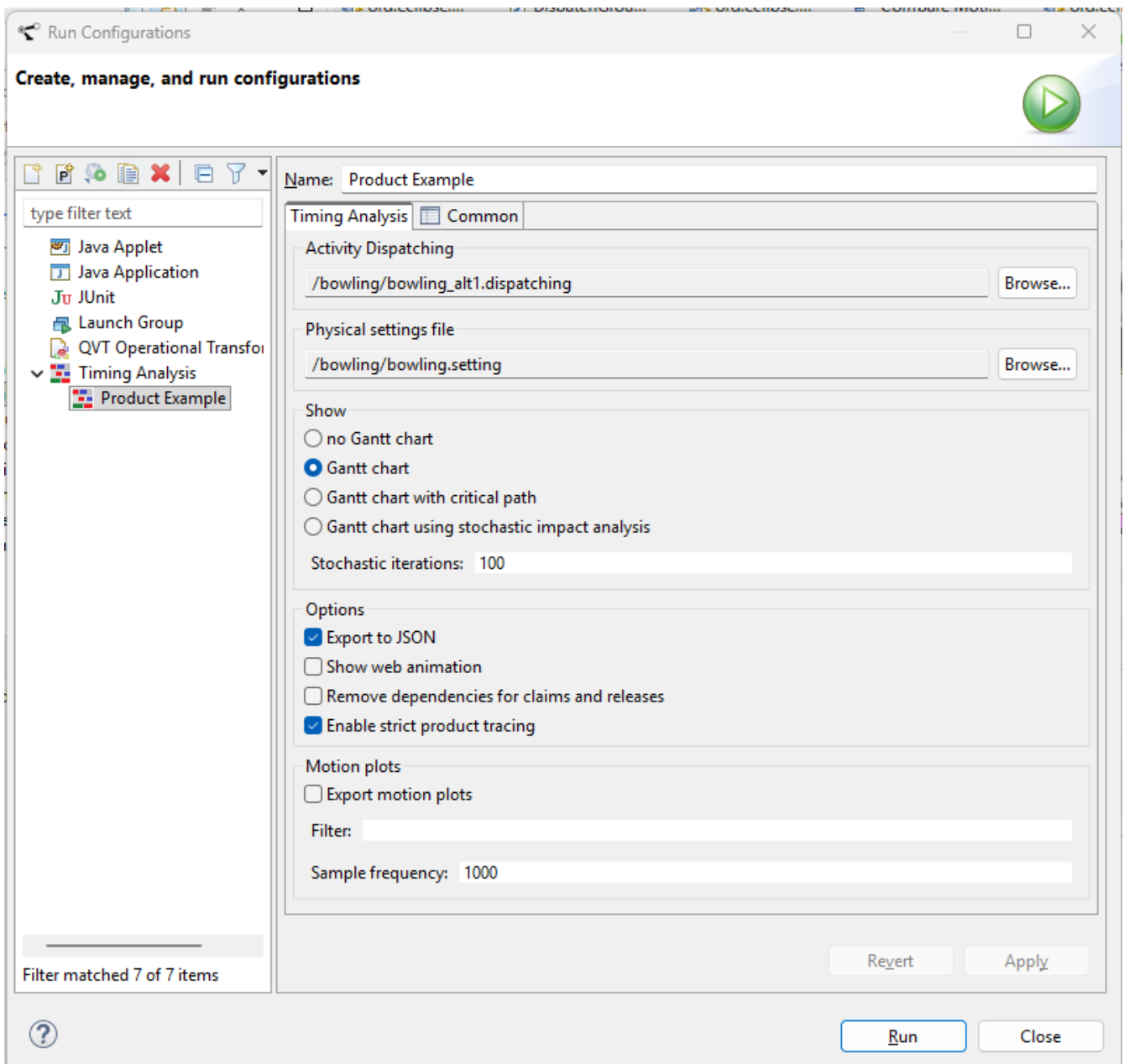


You can explore the effect of different settings by combining a dispatching file with different setting files.

For more advanced scheduling options you can use the *Run Configurations* window. Follow the above selection procedure but choose *Run As - Run Configurations...*, or use the dropdown from the green play button in the toolbar on top.



Available scheduling configurations show in the list on the left side of the *Run Configurations* window below the entry for *Timing Analysis*.



A new configuration can be created by using the *New Launch Configuration* button



A *Timing Analysis* run configuration consist of the following elements:

- **Name** Allows the user to identify the configuration for future usage.
- **Activity Dispatching** Each configuration refers to one particular *dispatching* file. This is a required field.
- **Physical settings file** The settings to be used to generate the schedule.
- **Show**
 - **no Gantt chart** No Gantt chart is generated
 - **Gantt chart** A Gantt chart is generated and is shown on-screen. The timing of the Gantt chart is based on fixed timing settings and default values. The coloring is based on correspondence between *Activities* and *Resources*, see below for a more detailed explanation.
 - **Gantt chart with critical path** A critical path analysis of the schedule is made, based on fixed timing settings and default values, and is represented in a Gantt chart. The coloring is based on the outcome of the critical path analysis. See below for an explanation of the coloring scheme.
 - **Gantt chart using stochastic impact analysis** Based on fixed timing and default values a Gantt chart is generated. The coloring is based on the probability that a task will be on the critical path given the specified timing distributions. In order to determine this probability 200 samples are drawn. For each sample a schedule is determined and a critical path analysis is performed. See below for an explanation of the coloring scheme.
- **Options**
 - **Export to JSON** Export the LSAT model and the results of the timing analysis to a JSON file.
 - **Show web animation** The scheduling produces skeleton files for *Javascript/Paperscript* based 2D animations. The default animation shows colored dots and by modifying the skeletons arbitrary animations can be produced. If the configuration also generates a Gantt chart an timing indicator is shown on the Gantt chart that is synchronized with the timing of the animation. The *Javascript* files are located at *analysis/scheduled/animated*.
 - **Remove dependencies for claims and releases** Before calculating the critical path the dependencies relating to claims and releases are cut short. As a result claims and releases are not included in the dependency graph and are thus removed from the critical path while preserving the net effective dependencies between actions. ***NB The claims and actions themselves are not removed from the schedule nor is the timing of the schedule influenced by this option.***
 - **Enable strict product tracing** Product tracing is always enabled and calculates the product flow of the system based on the schedule and the provided logistics specification. When *strict* mode is enabled, the scheduling will throw an error if the user has not modeled a sound product flow within the logistics specification. The product trace can be analyzed in *analysis/<name>.product-trace.txt*
- **Motion plots**
 - **Export motion plots** Include motion plots in the Gantt chart. This requires the motion calculator to export detailed information like position, velocity, acceleration, etc.
 - **Filter** Export only the motion plots of the peripherals that match the filter. For instance, *Robot.give.M Robot.take.M* only exports the motion plots of these two robots. Leave **Filter**

empty to export all motion plots.

- **Sample frequency** Set the sample frequency in hertz to which the motion data (position, velocity, acceleration, etc.) is down sampled before it is exported to the motion plots.

5.1.1. Product Trace Analysis

Product tracing is always enabled during timing analysis and generates a file named *<file>.product-trace.txt* in the *analysis/scheduled* folder. This file contains detailed information about the product flow of the system based on the schedule and the provided logistics specification.

- In **Run As (normal mode)** the product trace file contains:
 - Summary of the product flow through the system
 - List of any unprocessed product changes that could not be assigned to the modeled system structure
 - Statistics about product assignments and transfers
- In **Debug mode** the product trace file contains:
 - Complete trace of all decisions made during product assignment
 - Full details of every product change operation
 - All product-id assignments and their reasoning
 - Diagnostic information useful for validating the logistics specification

When *Enable strict product tracing* is enabled, the scheduling will throw an error if unprocessed product changes are detected, ensuring that the user has modeled a sound and complete product flow within the logistics specification.

5.2. Gantt Charts

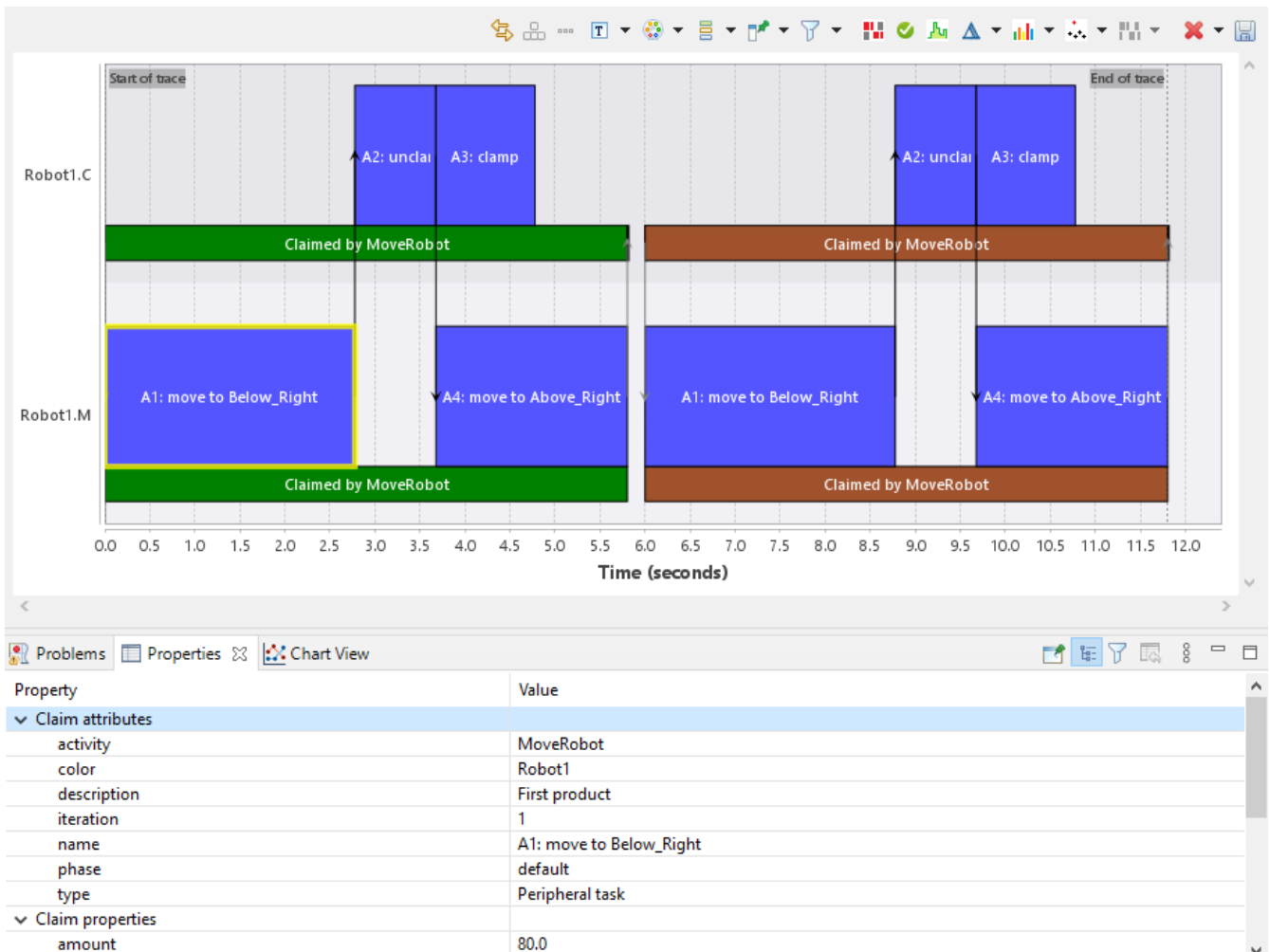
The Gantt chart shows a resource on the Y-axis for every Peripheral which participates in the activity sequence. The X-axis shows the relative time in seconds. Each resource in the Gantt chart shows a low bar and a high bar:

- | | |
|-----------------|---|
| Low bar | Represents the claiming of a <i>Resource</i> in the <i>Machine</i> by an <i>Activity</i> . This bar is replicated for all <i>Peripherals</i> in the claimed <i>Resource</i> . |
| High Bar | Represents an action that is performed by a <i>Peripheral</i> . An action can be either a move or an atomic action. |

The arrows represent the dependencies between the tasks.



When a Gantt chart is generated, result files are stored in the project folder *analysis/scheduled* (i.e. a Gantt file *<file>.etf*). The file name of the *etf* file is derived from the file name of the *dispatching* file that is scheduled. The Gantt file is automatically opened in an editor showing the Gantt chart.



The *Properties* view shows detailed information about the selected bar. user attributes, trace points, scheduling phases and repeat steps will also be shown in the properties and can be used to filter the Gantt chart.

5.2.1. Gantt chart

In case of a Gantt chart without critical path analysis the color of the low bars represent the *Activity* that holds the claim. Additionally all actions, the high bars, of the same *Resource* are colored equally. The black arrows indicate the dependencies as specified in the *Activity* and the gray arrows indicate the dependencies between *Activities* as derived from the claim/release strategy.

If a resource is passively claimed by multiple activities then the properties show the activities that apply. For passive claims the name shown is **Claimed passively by ...**

Scheduling the actions in the activities

An action can be scheduled using two types of algorithms: As Soon As Possible (ASAP) or As Late As Possible (ALAP). The default algorithm for all actions is **ASAP**, unless specified otherwise as described in [Logistics Specification](#). As mentioned previously, resource claiming or releasing are always **ASAP** by design.

ASAP executes each action on a resource as soon as the resource is available, and all preceding actions have finished.

ALAP executes each action on a resource as late as possible. You can see the difference between the

two schedules illustrated below.

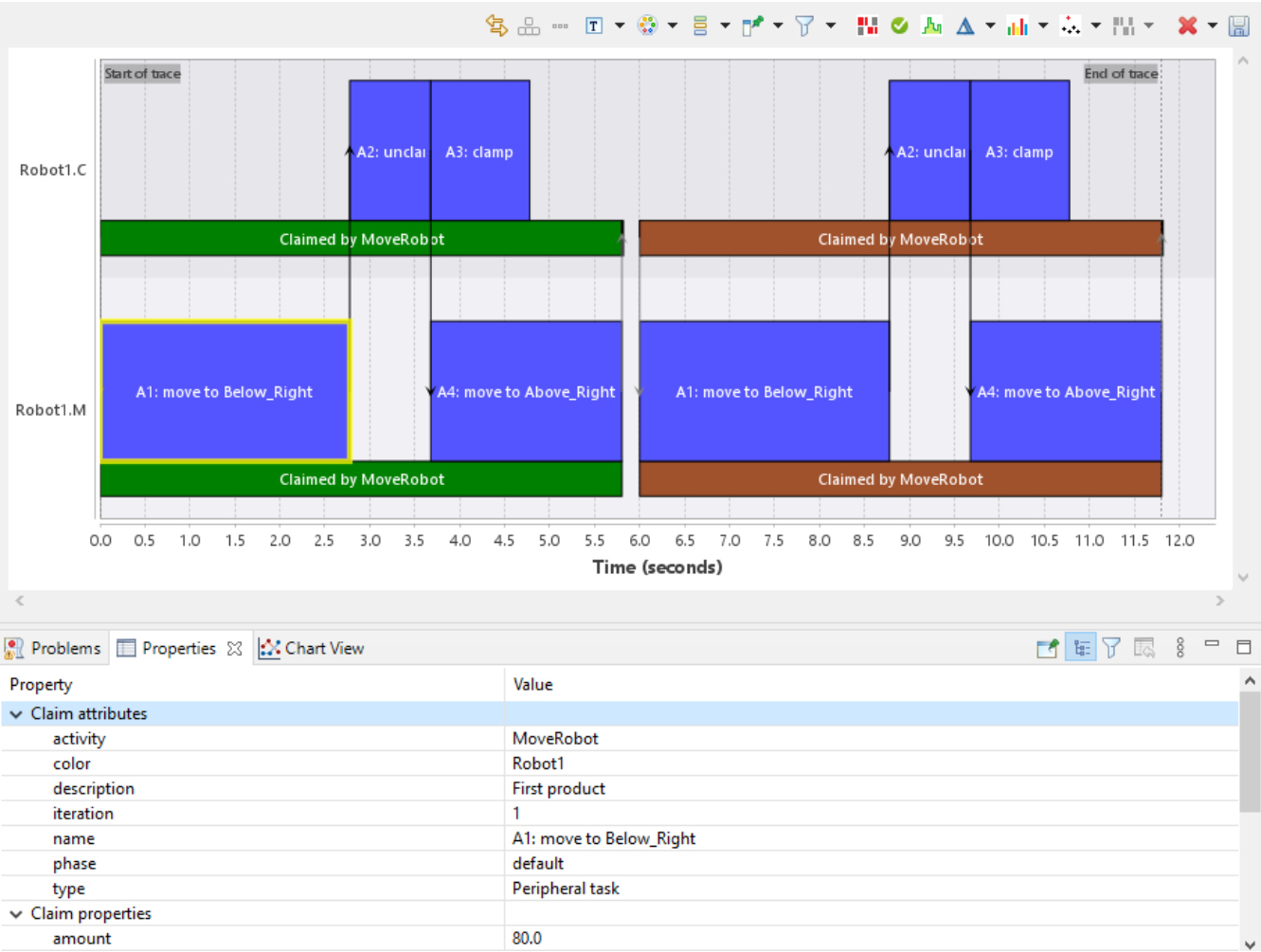


Figure 10. ASAP scheduling example for Action A3

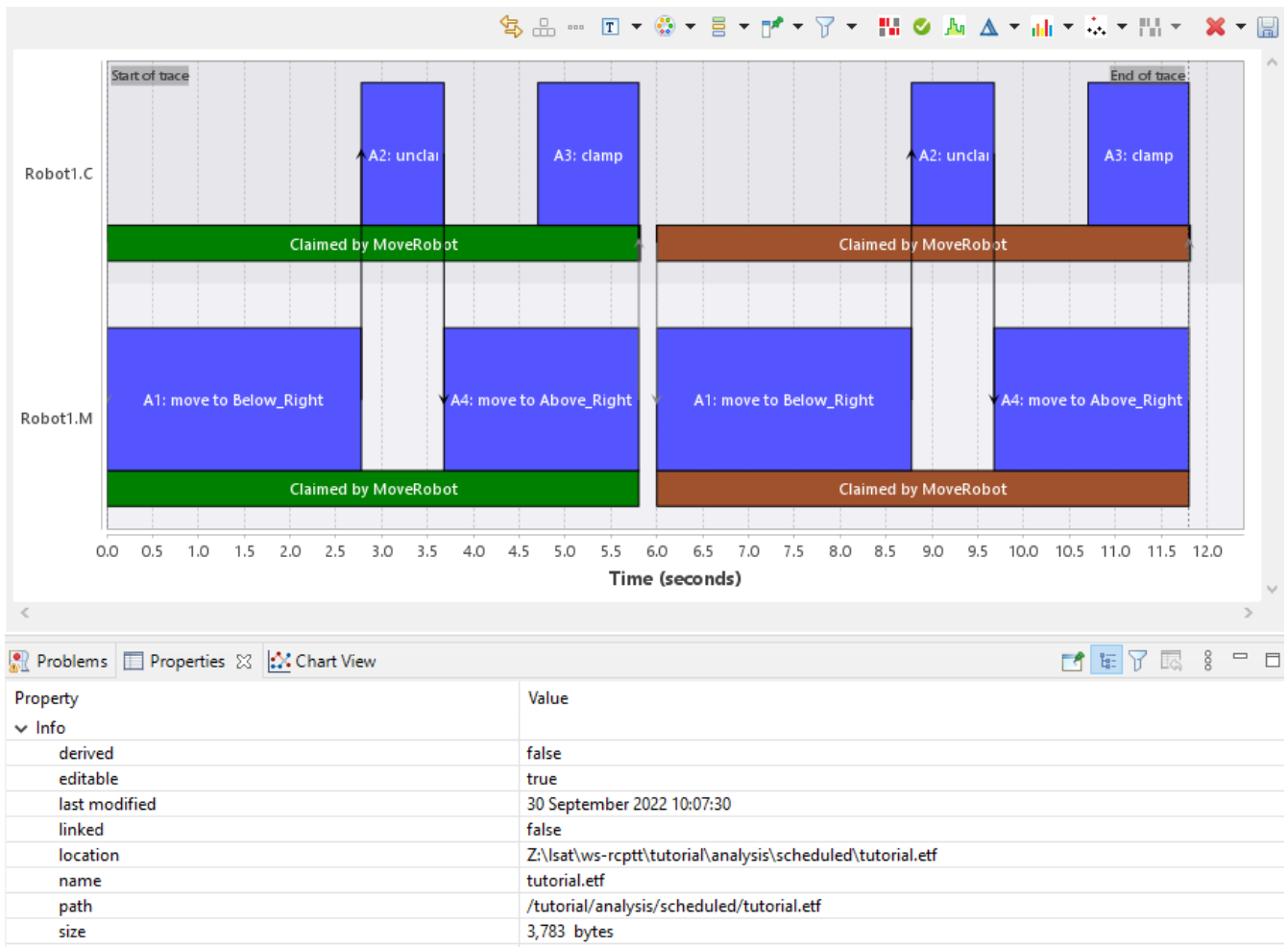


Figure 11. ALAP scheduling example for Action A3

Motion plots

Motion plots can be enabled in the Gantt chart via the hide/show button . A Gantt chart with motion plots shown is given below:

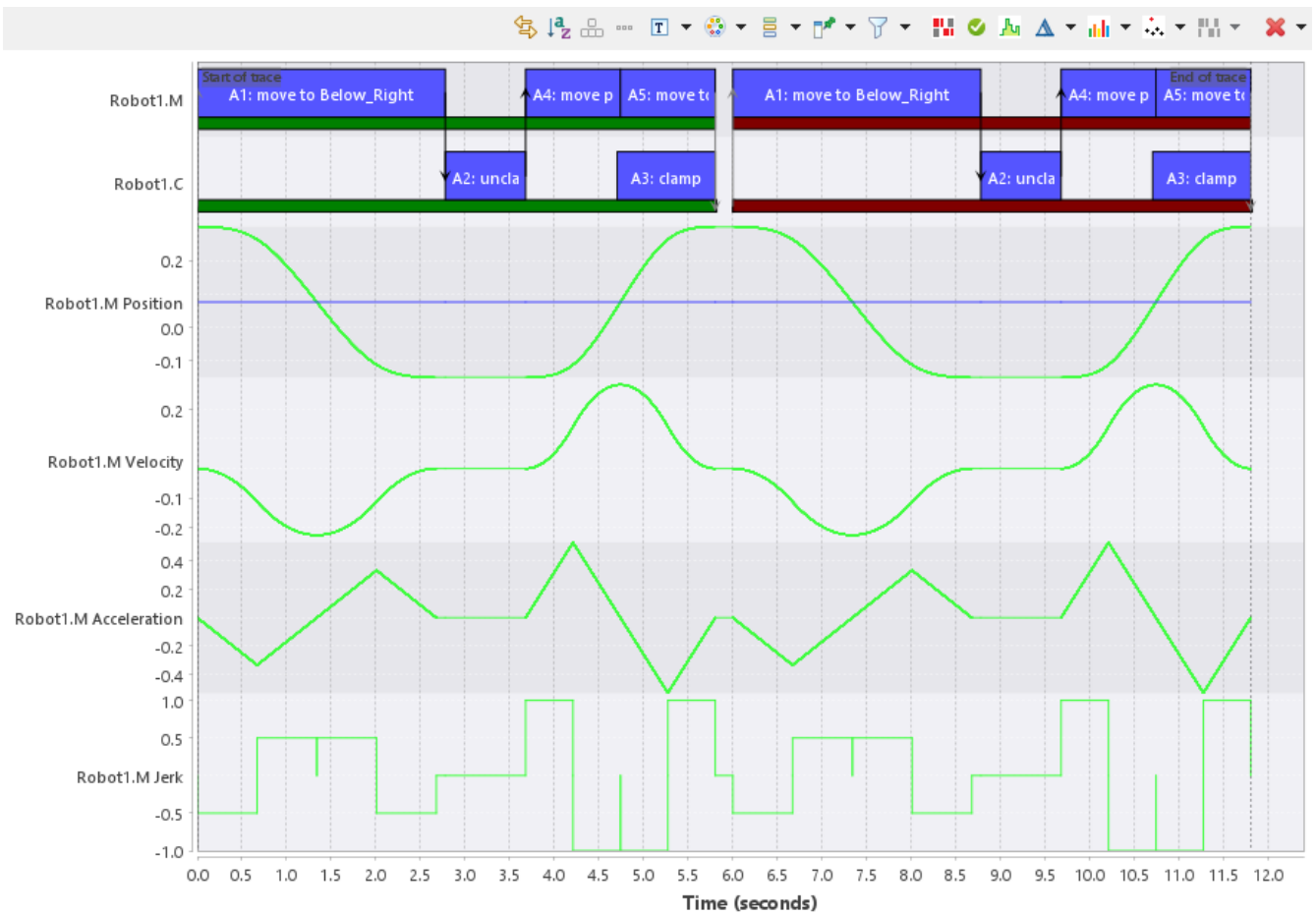


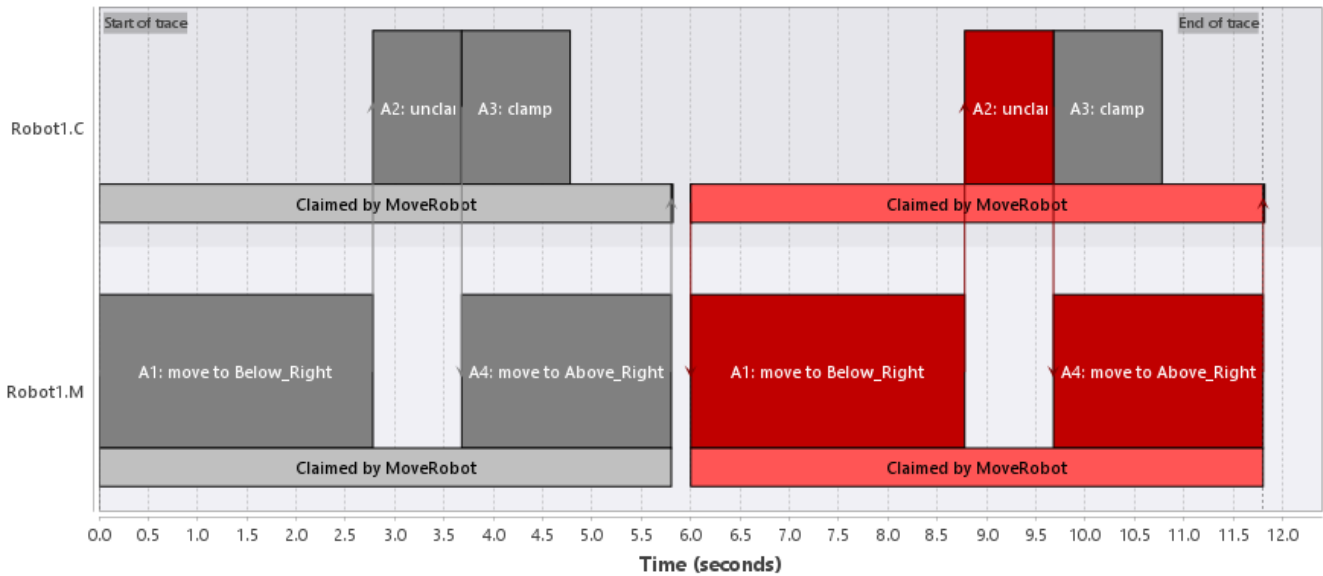
Figure 12. Gantt chart including motion plots showing the position, velocity, acceleration and jerk over time of Robot1

Note that only motion plots which are exported during the [timing analysis](#) can be plotted. Which of the exported motion plots are shown can be controlled by [filtering](#).

5.2.2. Gantt chart with critical path

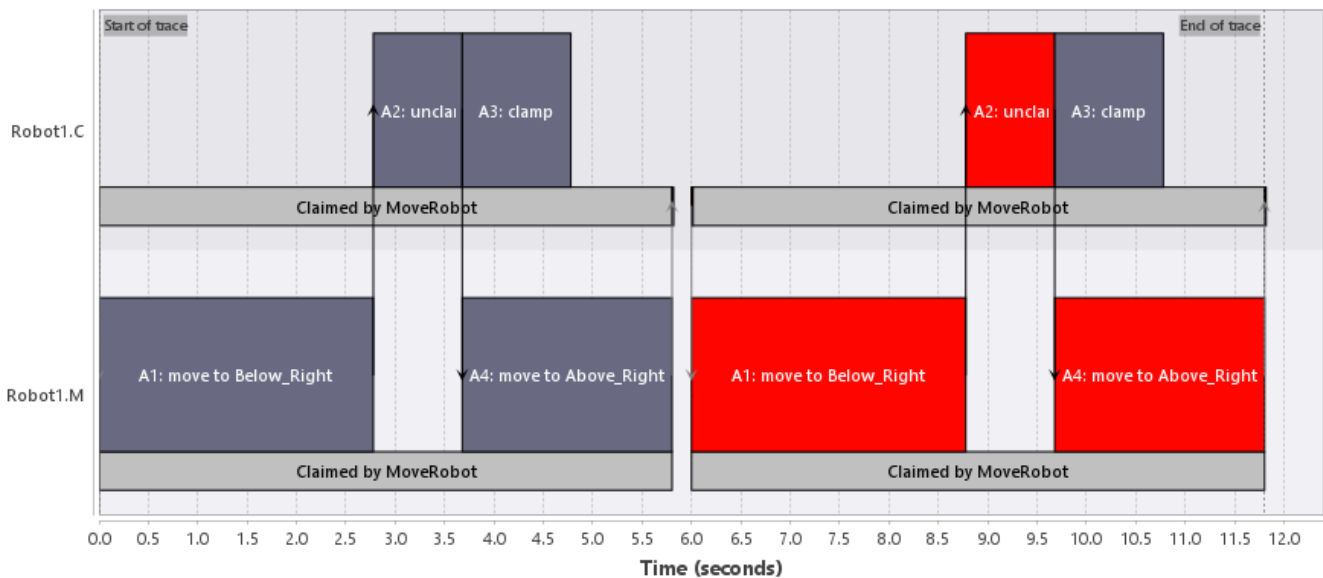
In project management the critical path is defined as “the sequence of scheduled activities that determines the duration of the project.” The same applies for the calculated schedule as visualized in the Gantt chart.

When a critical path analysis has been performed the actions and dependencies are either colored red or grey. Red indicates an action or dependency is on the critical path of the schedule. The low bars can be dark red, light red or grey. Dark red indicates both claim and release are on the critical path. Light red indicates either claim or release, but not both, are on the critical path. Grey is used to indicate neither claim or release are on the critical path.



5.2.3. Gantt chart using stochastic impact analysis

The actions in the Gantt chart of the stochastic impact analysis are colored based on the probability that the action is on the critical path taking into account the specified timing distributions. A dark bluish grey indicates that the action will never be on the critical path, bright red indicates the action is always on the critical path. Intermediate colors are used to indicate intermediate probabilities. Tasks can be selected by using the mouse and the Properties window shows the property *Critical* that displays the probability that the selected task is on the critical path.

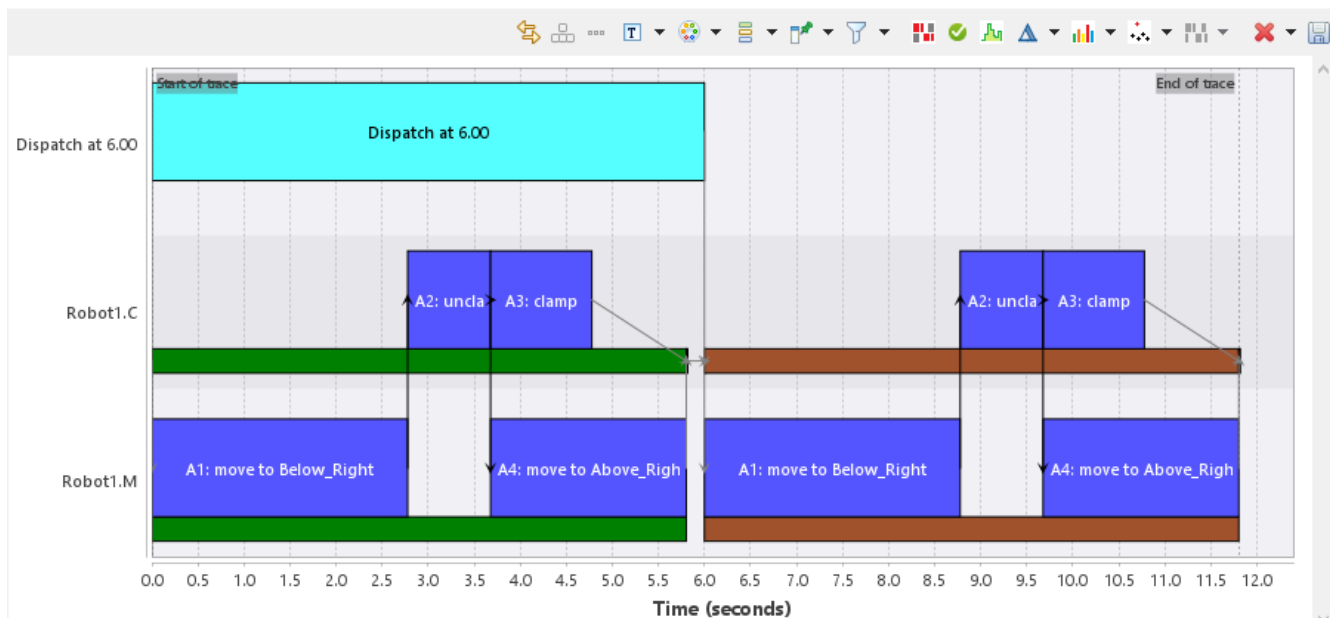


5.2.4. Filtering

By default the Gantt chart will hide all superfluous information. To show all information in the Gantt chart use the Eclipse TRACE4CPS toolbar to *Clear all filters*. Now all dependencies are drawn and also the dispatching offset is visualized.



For more information about the possibilities in the ESI Trace viewer, see the TRACE user manual in the Help menu.



Chapter 6. Throughput per Resource Editor

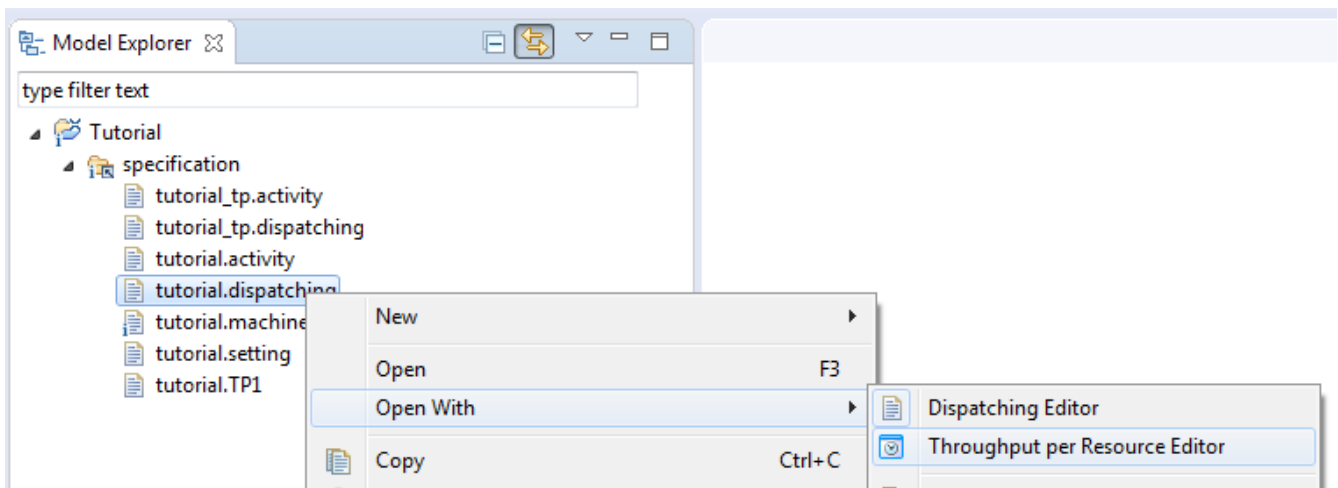
The throughput per resource editor helps to identify where the specification can be optimized for throughput performance of the system.

As the specification is partitioned in multiple files (i.e. dispatching, settings, schedule, etc.) this makes it rather difficult to identify where the specification can be optimized for throughput. For this a new view has been designed which provides an overview of the most critical information with respect to throughput optimization for a specific activity dispatching sequence.

For each resource in the specification a tab page is available in the editor. The tab page shows an ordered table with all actions that are relevant for its resource. More information about the table content can be found in section [The Resource Actions Table](#). The *Outline* view shows an overview of the throughput per resource and the throughput of the total activity dispatching sequence, for more information see section [The Outline View](#).

6.1. Opening the Throughput per Resource Editor

The Throughput per Resource editor shows information for a specific activity dispatching sequence. For this reason the editor can be opened by means of the context menu of a particular dispatching file. The use of the context menu is needed as by default the [Dispatching Editor](#) is opened when double-clicking a dispatching file. To open the Throughput per Resource editor right-click a dispatching file in the *Model Explorer* view and select *Open With* → *Throughput per Resource Editor* from the context menu.



Eclipse remembers the last editor used to open a file. When double-clicking the file this editor will be used again to open the file. The *Open With* context should be used to open the file using another editor.

6.2. The Resource Actions Table

The resource actions table contains a row for every action in the activity dispatching sequence that is relevant for the resource. The rows are ordered as such that their actions are ordered in topological order w.r.t. the activity dispatching sequence. Topological ordering basically means that

an action will never be listed before all actions required to be executed prior to the action. So this doesn't guarantee that two succeeding actions in the table are executed in sequence as they might be executed in parallel. For more information on how the action sequence is determined please see section [Calculating the Throughput per Resource](#).

In general a table cell will be gray when its information is not applicable for the action at that table row. When needed the data in the table either can be refreshed or reloaded.

Refreshing the data - by means of the 🔄 toolbar button - will update all data in the resource actions table and *Outline* view reflecting the current state of the editor including already made changes.

Reloading the data - by means of the 🔄 toolbar button - will discard all made changes and reload the specification models. This action can be used to reload changes in the specification which are made outside the Throughput per Resource editor.



The Throughput per Resource editor will not detect added/deleted resources, axes and setpoints. When these are modified the Throughput per Resource editor needs to be closed and re-opened again.

The table contains the following columns:

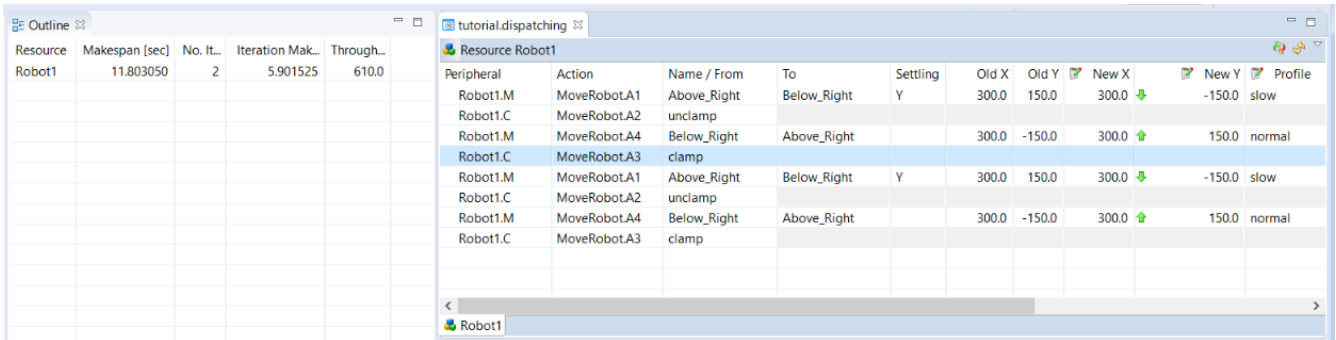
Peripheral	The fully qualified name (format <i><Resource Name>.<Peripheral Name></i>) of the peripheral performing the action.
Action	The fully qualified name (format <i><Activity Name>.<Action Name></i>) of the action.
Name / From / Distance	Either the name of the peripheral action, the start position or the distance of the move.
To	The end position for the move (if applicable).
Settling	The axes that require settling for the move.
Old <Axis>	The start location per axis for the move.
New <Axis>	The end location per axis for the move. An icon 📏 indicates that the axis is adjusted as part of the move. The 🛠 icon in the column header indicates that the values in this column may be edited.
Profile	The motion profile name for the move. The 🛠 icon in the column header indicates that the values in this column may be edited.
<Setpoint> time	For every setpoint, the minimum time needed to adjust the setpoint using its axis motion profiles. In other words, the time needed to adjust the setpoint conforming to its motion profile, not taking into account the adjustment of other setpoints, thus no stretching will be applied.

Time

The time needed to perform the action. An icon  indicates that the action is on the critical path of the resource. For more information see section [Calculating the Throughput per Resource](#) and section [Gantt chart with critical path](#). The  icon in the column header indicates that the values in this column may be edited. Do note though that this only applies for the times of peripheral actions and not for moves.



When editing: If the setting file contains a composed expression it will be replaced with the edited constant value.



The screenshot shows two windows. The 'Outline' window on the left displays a summary table for Resource Robot1. The 'tutorial.dispatching' window on the right shows a detailed table for the same resource.

Resource	Makespan [sec]	No. It...	Iteration Mak...	Through...
Robot1	11.803050	2	5.901525	610.0

Peripheral	Action	Name / From	To	Setting	Old X	Old Y	New X	New Y	Profile
Robot1.M	MoveRobotA1	Above_Right	Below_Right	Y	300.0	150.0	300.0	-150.0	slow
Robot1.C	MoveRobotA2	unclamp							
Robot1.M	MoveRobotA4	Below_Right	Above_Right		300.0	-150.0	300.0	150.0	normal
Robot1.C	MoveRobotA3	clamp							
Robot1.M	MoveRobotA1	Above_Right	Below_Right	Y	300.0	150.0	300.0	-150.0	slow
Robot1.C	MoveRobotA2	unclamp							
Robot1.M	MoveRobotA4	Below_Right	Above_Right		300.0	-150.0	300.0	150.0	normal
Robot1.C	MoveRobotA3	clamp							

6.3. The Outline View

The image in the previous section also shows the *Outline* view. This view shows a table with an overview of the throughput per resource and - in the last row - the throughput for the total activity dispatching sequence. For more information on how the throughput is calculated for a resource, please see section [Calculating the Throughput per Resource](#).

The table contains the following columns:

Resource	The resource for which the throughput is calculated.
Makespan	The total time needed to perform all resource actions in the activity dispatching sequence.
No. Iterations	The number of iterations for this resource in the activity dispatching sequence, for more information see section Specifying the number of iterations .
Iteration Makespan	The total time needed to perform exactly 1 iteration in the activity dispatching sequence.
Throughput	The theoretical maximum number of iterations per hour that this resource is capable of executing.

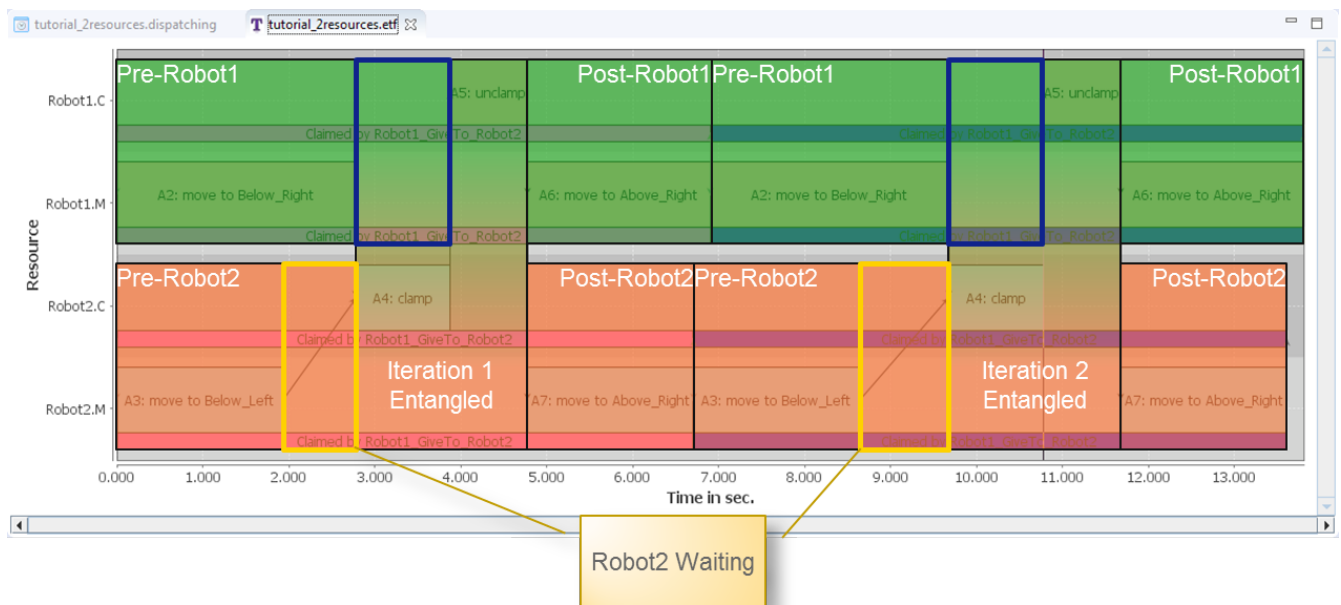
6.4. Calculating the Throughput per Resource

This section provides some background information on what is meant by calculating the throughput per resource.

Though the systems throughput performance is determined by the full orchestration of all actions for all resources and peripherals in the system, sometimes it is good to know which resource is most critical in this orchestration. When calculating the throughput per resource the theoretical maximum throughput of the resource is calculated as if no actions of other resources are limiting its throughput, with the exemption of all actions in the 'entangled' section of an activity.

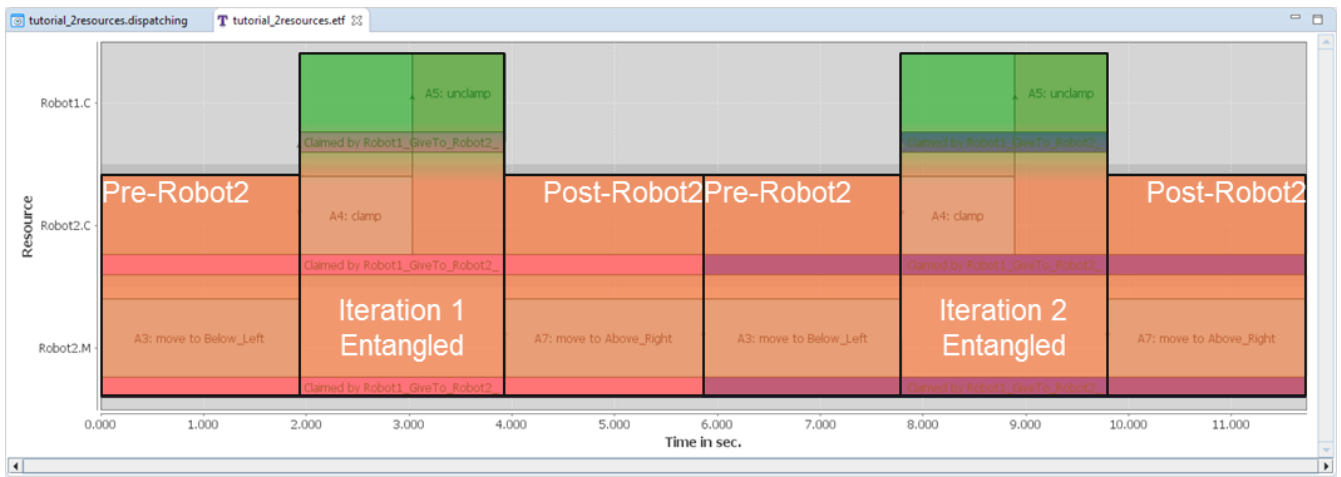
The entangled section of an activity

With the entangled section of an activity the section is meant in which multiple resources are working together in harmony, most likely to transfer a product from one location to another. Maybe this can be explained a bit easier by use of the following example. The figure below shows the schedule of an activity dispatching sequence with an overlay that illustrates the pre, post and entangled sections of an activity. The activity 'Robot1_GiveTo_Robot2' is repeated twice in the dispatching.



In the entangled section Robot2 first needs to clamp before Robot1 is allowed to unclamp. This causes a gap in the schedule of Robot1 as illustrated by the blue rectangles. As these gaps are within the entangled section, they are kept in the schedule when calculating the throughput for Robot1.

Additionally to gaps in the entangled section, Robot2 also has gaps in the schedule in its prepare section, as illustrated by the yellow rectangles in the schedule. These gaps are caused by Robot1 delaying Robot2. For calculating the throughput for Robot2 the pre and post sections of Robot1 will be omitted and a schedule will be used as illustrated in the following image.



Chapter 7. Generating an Optimal Activity Dispatching Sequence

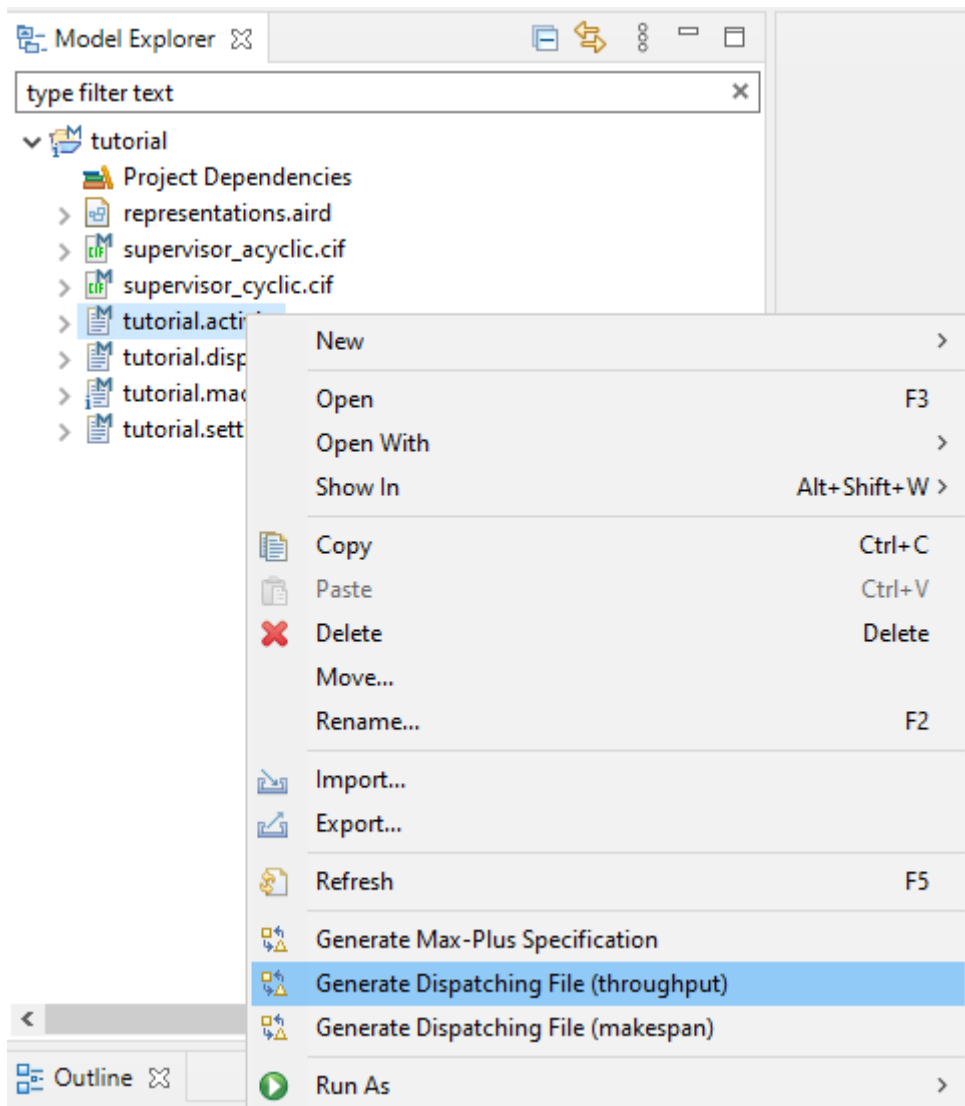
A supervisory controller modeled using CIF, see [Create Supervisory Controller Specification](#), captures all possible activity dispatching sequences. Given such a specification, an optimal activity dispatching sequence can be computed. We distinguish between two situations:

1. **Optimal continuous processing of products** In the first case, the supervisory controller specifies all possible activity sequences to establish a continuous product flow. The supervisor should have cycles, which represent repeatable activity sequences, modeling the continuous processing of products. An optimal activity dispatching sequence can be generated, that contains a repeatable activity sequence achieving the maximum throughput, as well as the shortest transient activity sequence to get to this steady-state repeatable activity sequence from the initial system state.
2. **Optimal processing a batch of products** In the second case, the supervisory controller specifies all possible product flows for a batch of products. If marked states are not considered as target states, it is assumed that any path to a state without enabled edges means that the product flow successfully completed. If marked states are considered as target states, then those states indicate a successful completion. An optimal activity dispatching sequence can be generated that achieves the minimum makespan, i.e., the fastest way to execute all required activities and reach a final state.

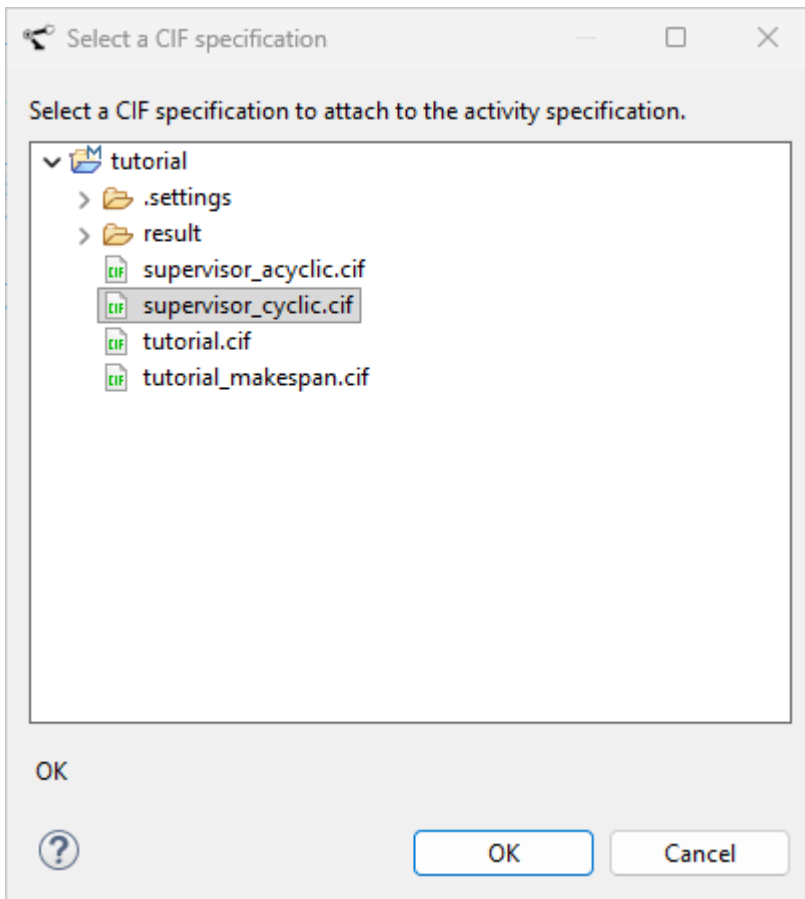
7.1. Generating a Dispatching File for Maximum Throughput

Target element: an activity specification

Right click on a target element, select *Generate Dispatching File (throughput)* to generate an activity dispatching sequence with maximum throughput.



Select a relevant CIF specification file.



In this example, we use the following CIF specification.

```
controllable MoveRobot;  
  
supervisor Tutorial:  
  
  location l0:  
    initial;  
    edge MoveRobot goto l1;  
  
  location l1:  
    edge MoveRobot goto l0;  
  
end
```

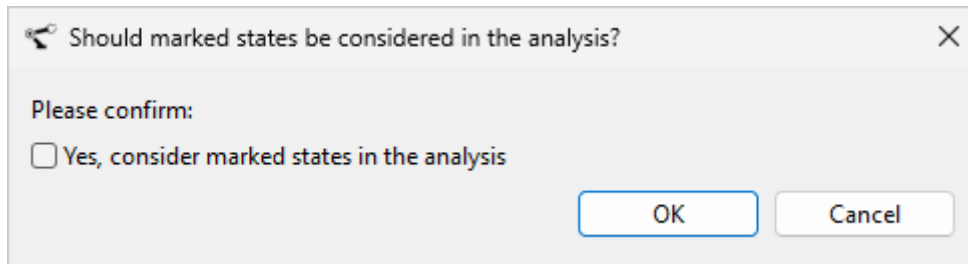
Click 'OK' to load the CIF specification.

Decide whether marked states should be considered in the throughput analysis. States that are indicated as **marked** in the CIF specification are considered as indicating a successful (partial) completion. For example, marked states can be used to indicate when a finished product has leaved the system.

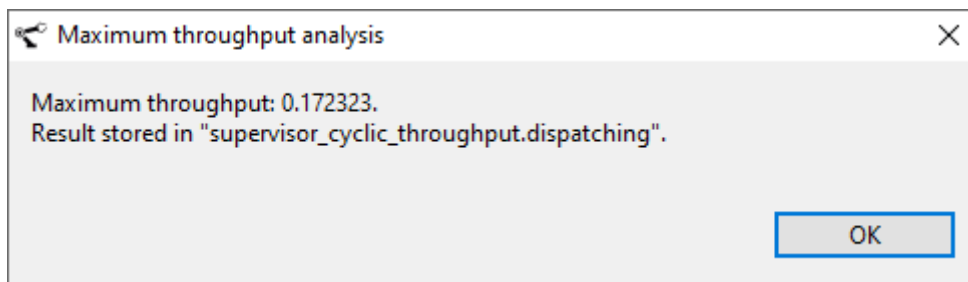
- If marked states are used, then the algorithm only considers cycles that have at least one marked state. Within this set of cycles, the algorithm finds the cycle that represents a repeatable activity sequence that achieves the maximum throughput.

- If marked states are not used, then all cycles are considered.

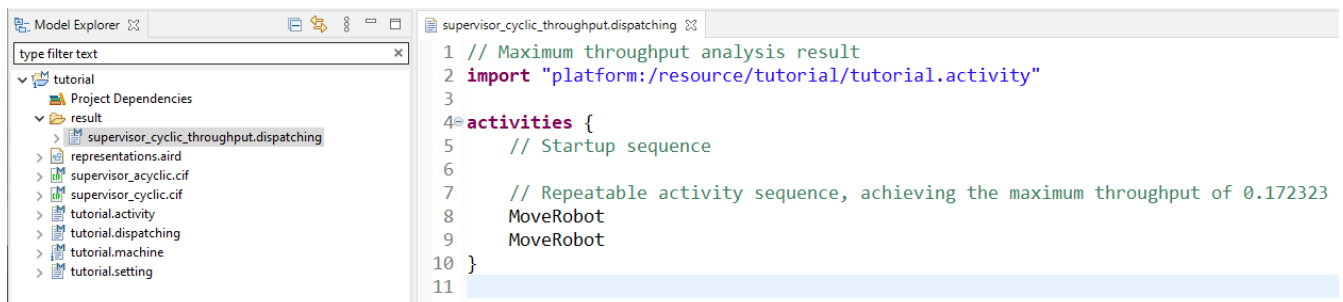
In our example model we do use marked states. Therefore we do not select the checkbox. Click 'OK' to start the analysis.



Once the analysis is finished, the computed throughput value is reported.

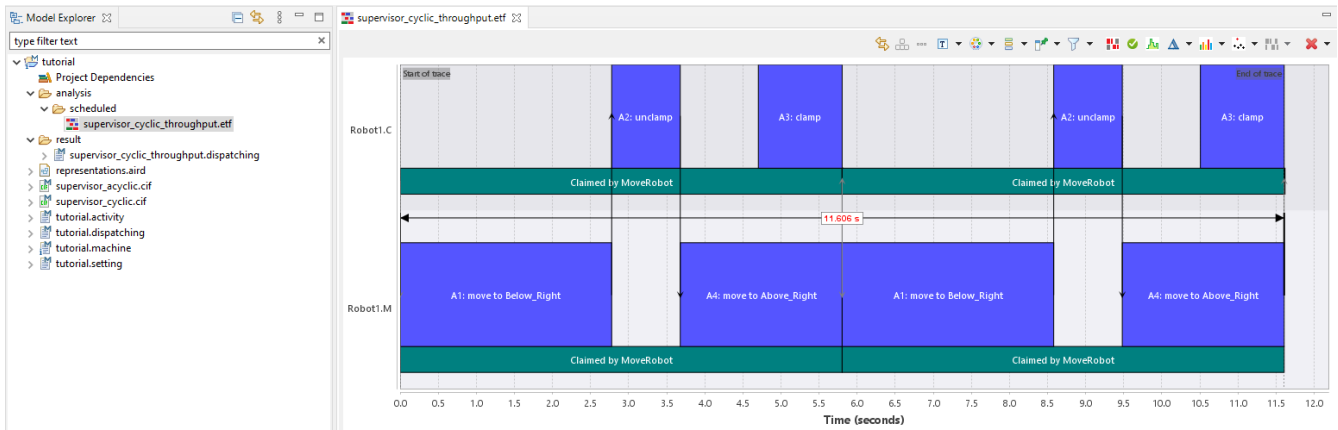


The dispatching file containing the optimal activity dispatching sequence is generated in the *result* folder.



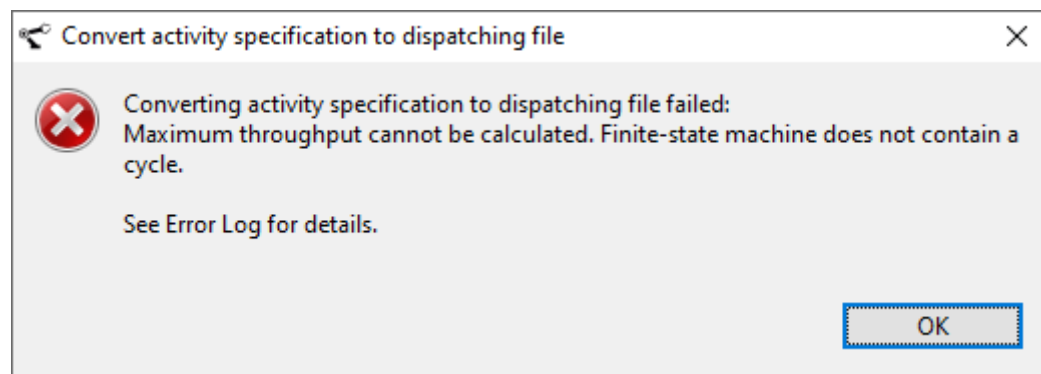
The first part of the dispatching file specifies the activity sequence that is needed to reach the steady state of the system. In the steady state, the same repeatable activity sequence can continuously be executed. In our example, this steady state is immediately reached. The throughput is defined as the number of activities that are executed, divided by the total execution time of the cycle. In our example, the throughput is 2 activities / 11.606 sec = 0.1723 activities per second.

This can visually be shown in the following corresponding Gantt chart.



To schedule the generated activity dispatching sequence in the form of a Gantt chart, see section [Schedule Activities](#).

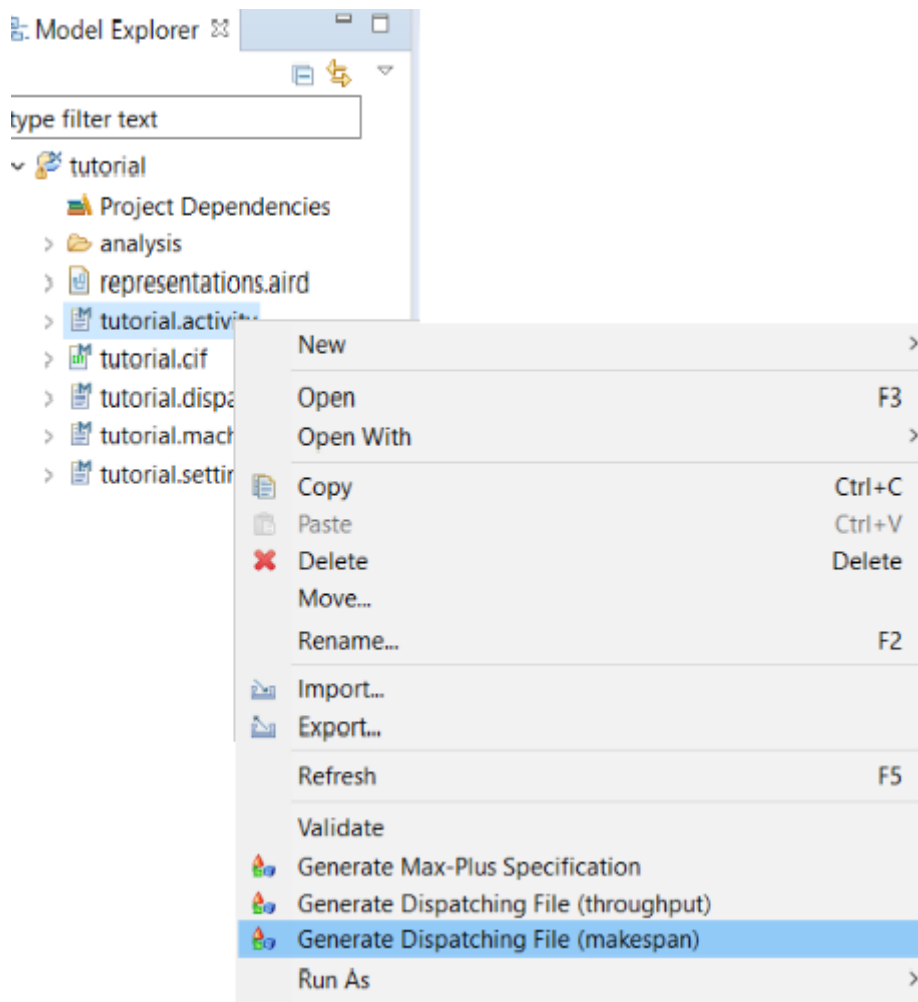
To calculate the maximum throughput, the CIF file must contain a cycle.



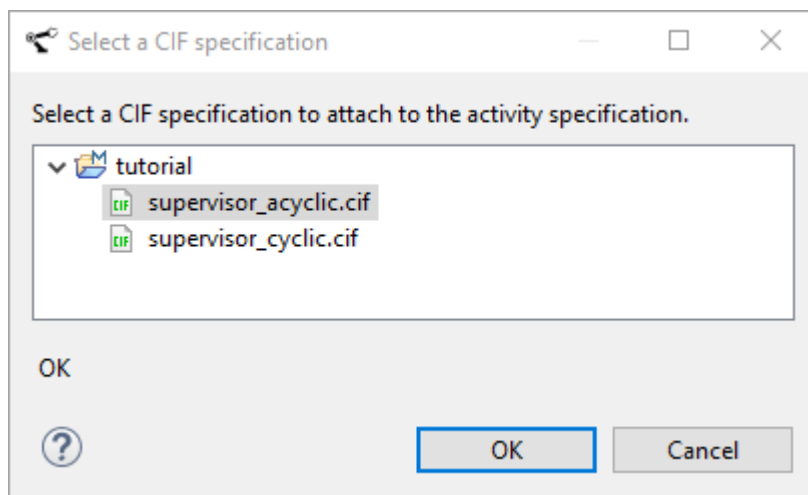
7.2. Generating a Dispatching File for Minimum Makespan

Target element: an activity specification

Right click on a target element, select *Generate Dispatching File (makespan)* to generate an activity dispatching sequence with minimum makespan.



Select a relevant CIF specification file.



In this example, we use the following CIF specification.

```
controllable MoveRobot;

supervisor tutorial:
    location l0:
        initial;
        edge MoveRobot goto l1;
```

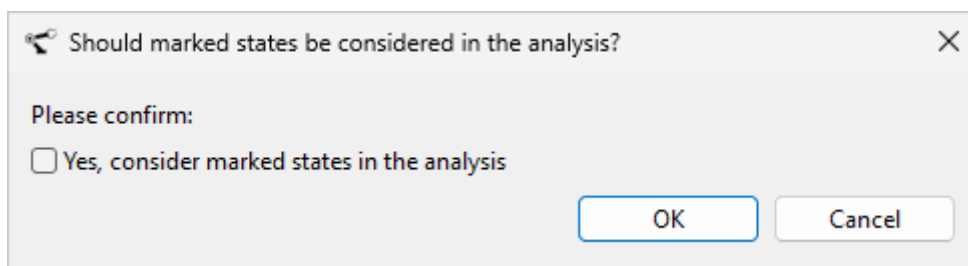
```
location l1:
    edge MoveRobot goto l2;
location l2:
    marked;
end
```

Click 'OK' to load the CIF specification.

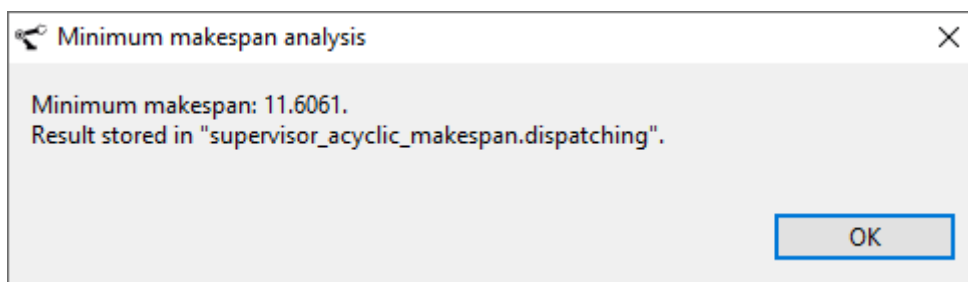
Decide whether marked states should be considered in the makespan analysis. States that are indicated as **marked** in the CIF specification are considered as target states indicating a successful completion. For example, marked states can be used to indicate when the system has produced a batch of products.

- If marked states are used, then the algorithm only considers activity sequences from the initial state to a marked state. The algorithm computes the shortest path (having minimum makespan) from the initial state to any of the marked states in the CIF model.
- If marked states are not used, then all activity sequences from the initial state to any state without outgoing edges are considered.

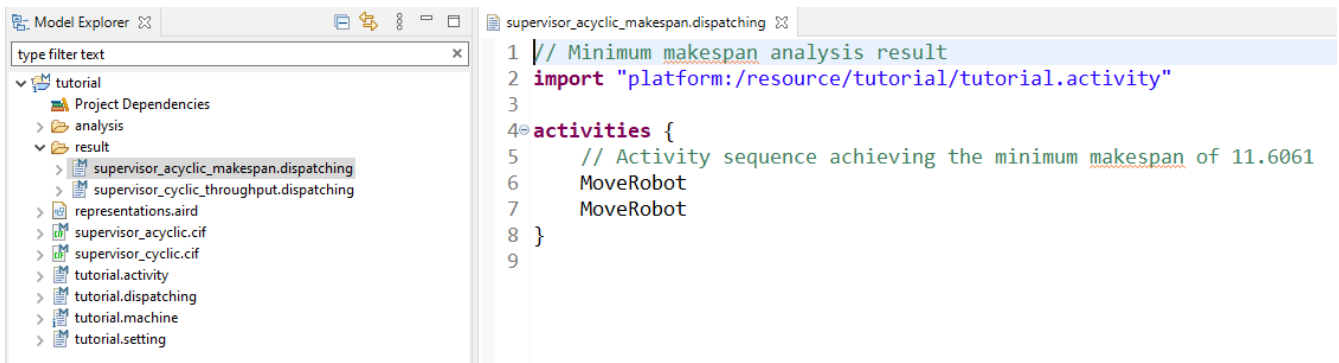
In our example model we do use marked states. Therefore we do not select the checkbox. Click 'OK' to start running the makespan analysis.



Once the analysis is finished, the computed makespan value is reported, which is the total execution time of the activity sequence.

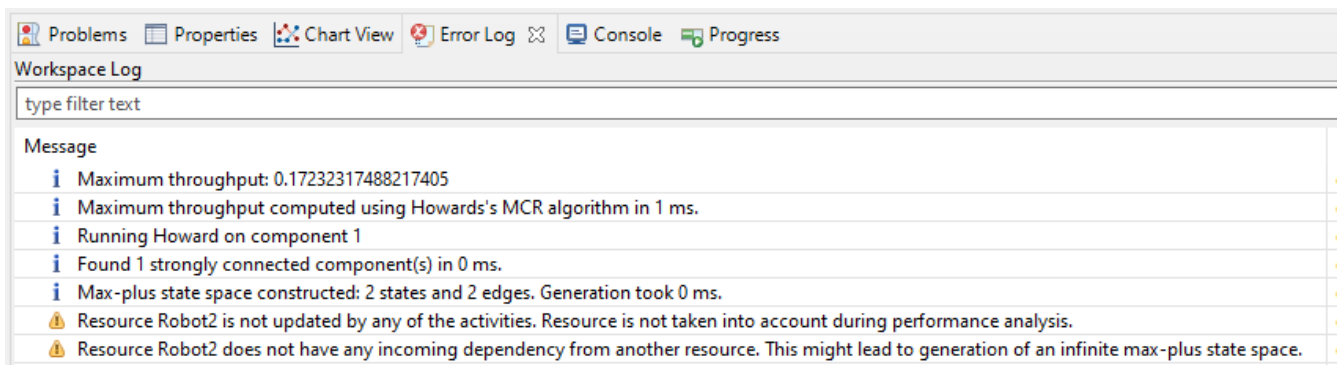
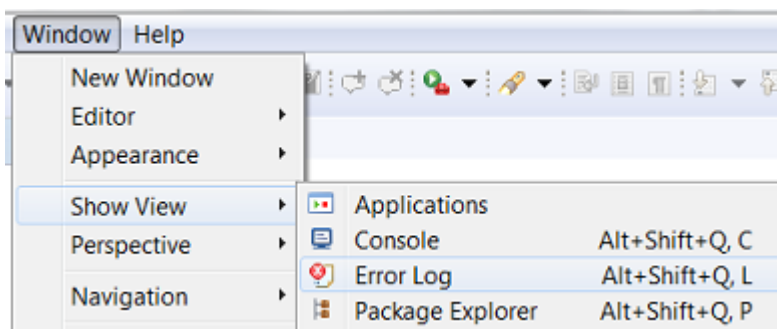


The dispatching file showing the optimal activity dispatching sequence is generated in the *result* folder.



7.3. Viewing Makespan/Throughput Values

The minimum makespan/maximum throughput, as well as some details about the algorithm execution, can be seen in the *Error Log* view. The *Error Log* view can be opened by clicking *Window* in the toolbar on top and *Show View* → *Error Log*.



An infinite state space can be generated during the throughput analysis, if there are multiple resources that can execute actions without ever synchronizing. A warning is generated if a resource is nowhere used in the analysis, or if it is never synchronized with another resource. To prevent a never terminating analysis, these problems should be resolved in the specification.

Chapter 8. Conformance Check

Conformance checking functionality is used to check whether traces from log files are able to be projected to a certain dispatching.

8.1. Add Trace Point

Before conducting a conformance check, trace points are supposed to be added to activity specifications. Trace points, logged in *trace* log files, are used to identify the starting and end points of an event. An example of log file is given below. Each line in a log file contains all the information of time and a trace point.



By default, log files with an extension `trace` are supported for conformance checking. A trace file is line based where each line starts with an instant in UTC (i.e. parsed using `DateTimeFormatter.ISO_INSTANT`), followed by comma-space and then the trace point.

```
2011-12-03T07:15:30.454994Z, TP-0001
2011-12-03T07:35:15.454894Z, TP-0002
2011-12-03T07:35:15.455168Z, TP-0005
2011-12-03T07:35:15.455172Z, TP-0006
2011-12-03T07:35:15.455429Z, TP-0007
2011-12-03T07:35:15.455434Z, TP-0008
2011-12-03T07:35:15.455588Z, TP-0009
2011-12-03T07:35:15.455591Z, TP-0011
2011-12-03T07:35:15.455797Z, TP-0010
2011-12-03T07:35:15.455805Z, TP-0012
2011-12-03T07:35:15.456519Z, TP-0003
2011-12-03T07:35:15.456716Z, TP-0004
```

In the first line, “TP-0001” is the string identifying the trace point in principle which is unique throughout the entire log files.

Additionally, there must be a trace point specification, which maps a trace point to either a starting or an end point of a known action. For instance, *TP-0001* maps to *start claiming Robot*. In order to perform conformance check, trace points are supposed to be added to its related activity specification surrounding an action definition.

```
import "tutorial.machine"

activity MoveRobot {
  prerequisites {
    Robot1.M at Above_Right
  }
  actions {
    C: ["TP-0001"] claim Robot1 ["TP-0002"] ①
    R: ["TP-0003"] release Robot1 ["TP-0004"]
  }
}
```

```

A1: ["TP-0005"] move Robot1.M to Below_Right with speed profile slow ["TP-0006"]
A2: ["TP-0007"] Robot1.C.unclamp ["TP-0008"]
A3: ["TP-0009"] Robot1.C.clamp ["TP-0010"]
A4: ["TP-0011"] move Robot1.M to Above_Right with speed profile normal ["TP-0012"]
}
action flow{
  C -> A1 -> A2 -> |S1 -> A3 -> |S2 -> R
                        |S1 -> A4 -> |S2
}
}

```

① ["TP-0001"] and ["TP-0002"] represent starting and end points for claiming a Robot.

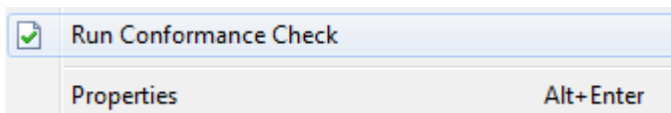
8.2. Conduct Conformance Check

After adding all the trace points to an activity specification, conformance check can be applied upon a dispatching file.



Dispatching offset is not supported for conformance check currently.

Right click on a dispatching file and select *Run Conformance Check*.



Note that in order to obtain a successful conformance check based on the above activity file, the dispatching file look like:

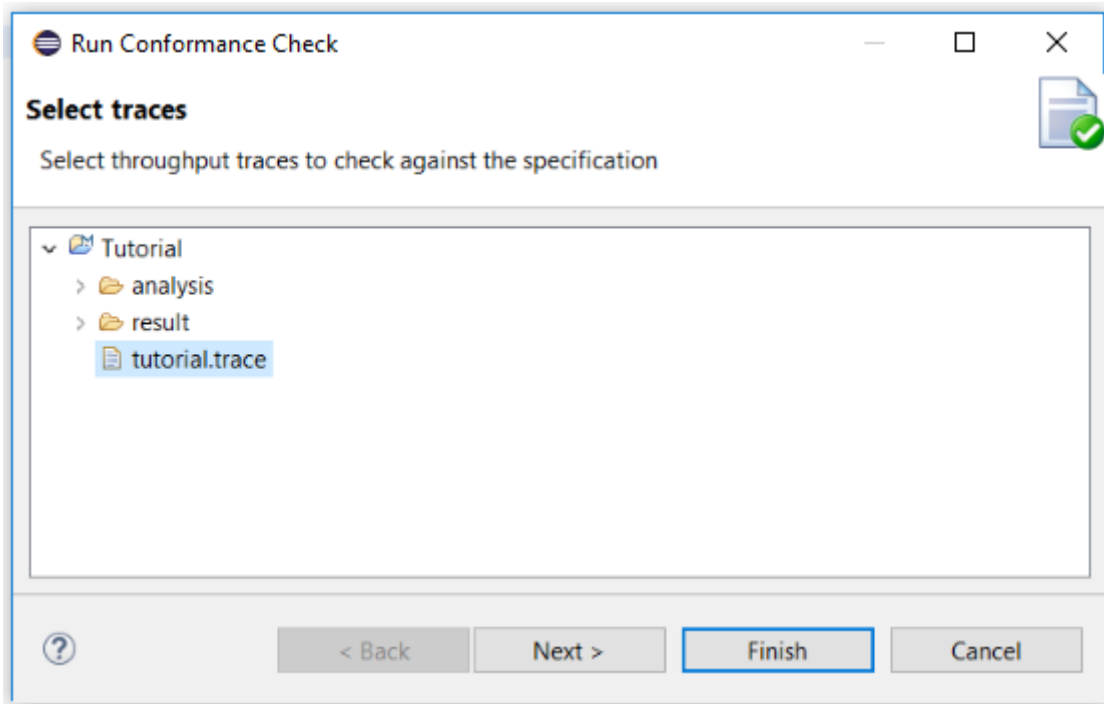
```

import "tutorial_tp.activity"

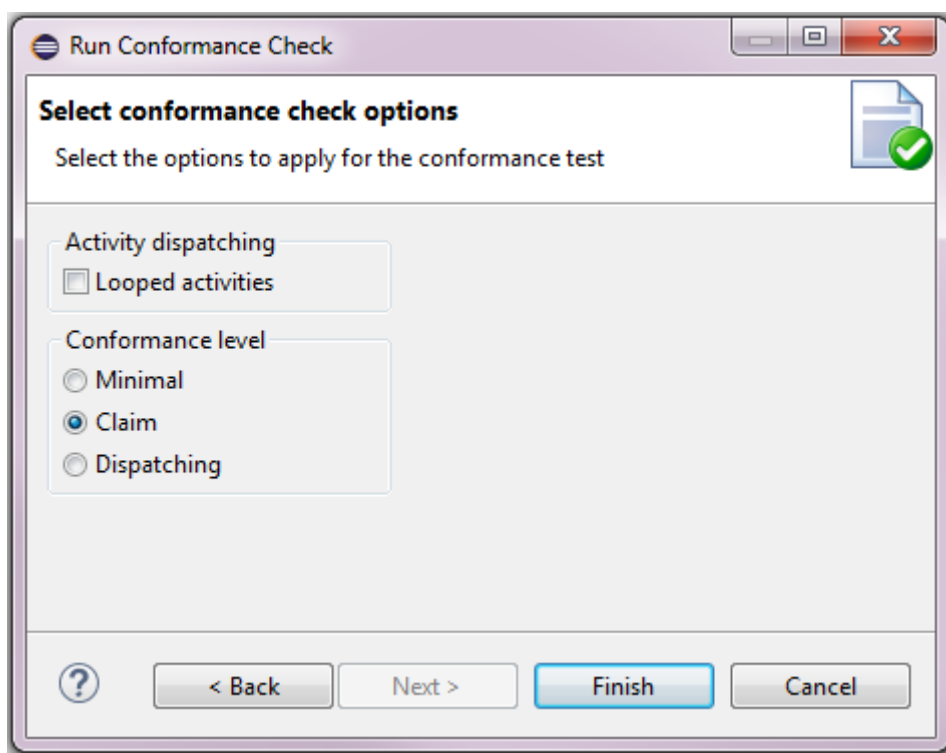
activities {
  MoveRobot ("First product")
}

```

Select *trace* files to run the conformance check, and *Next*.



There are several configurations for conformance check.

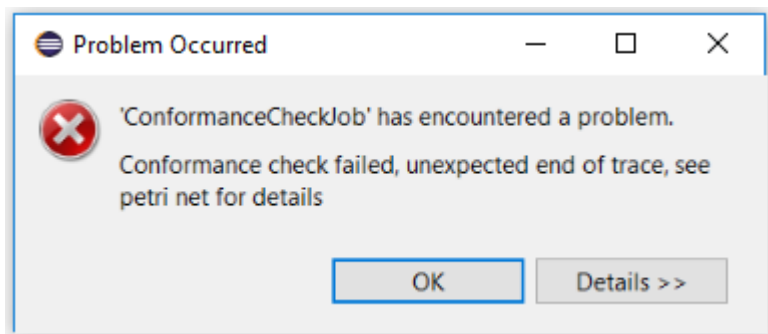
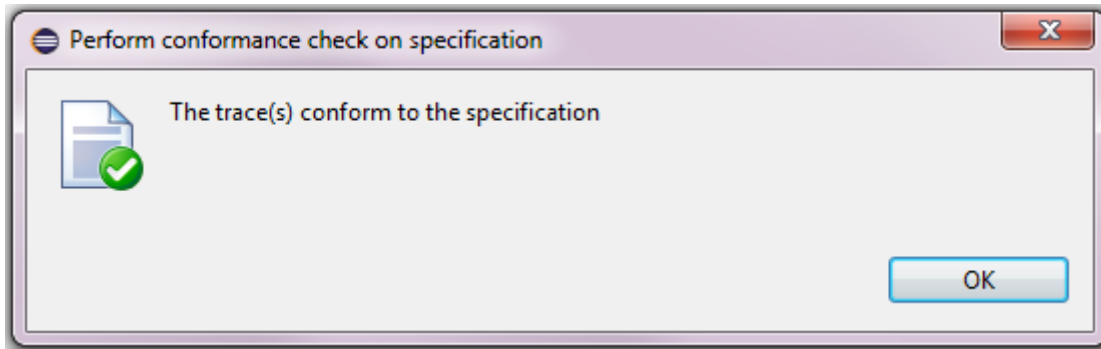


Activity dispatching Selected, if it activity dispatching occurs multiple times in log files.

Conformance level There are three levels for conformance check.
Minimal - only recognized trace points are kept for conformance check. It will pass, as long as it find one valid projection.
Claim - based on minimal level, all trace points for the claimed resources are used for conformance check.
Dispatching - most strict conformance check, it requires all the recognized trace points in log files are perfectly matched, even before claiming and after releasing a resource.

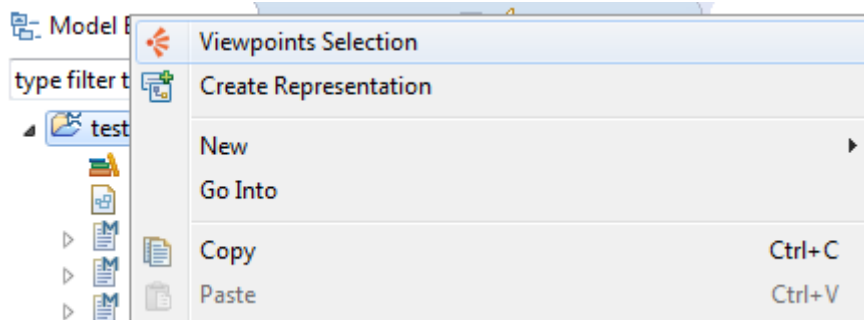
Set conformance check options and *Finish*. The conformance check will be executed.

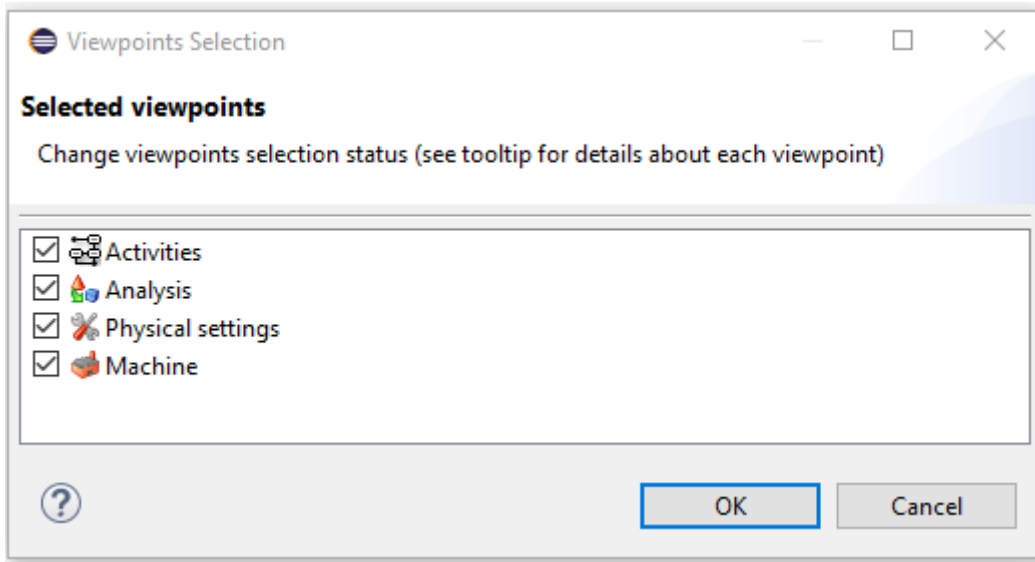
After the progress is completed, a pop-up will appear to show whether or not it passes the conformance check.



If a conformance check fails, a *petrinet* file will be generated under the folder *analysis/conformance*. It is used to diagnose conformance check failure.

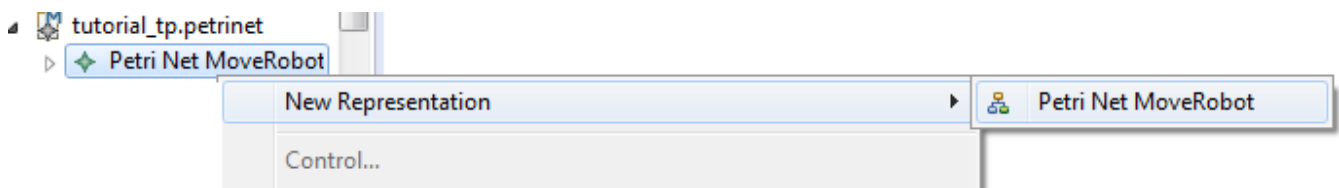
In order to open a graphical view of a Petri Net, a new viewpoint needs to be added to the project.



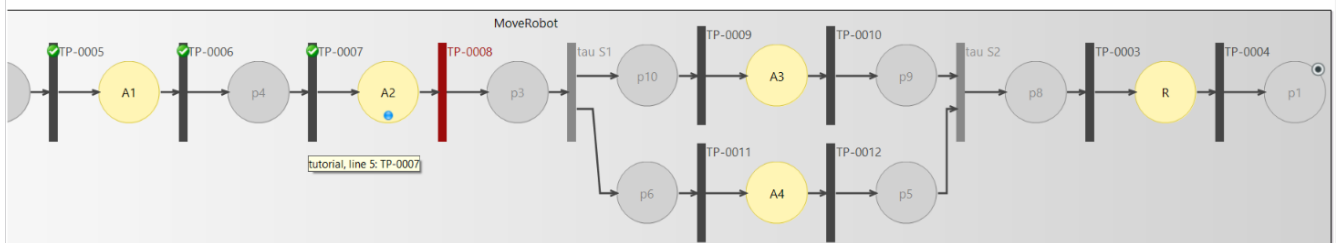


If the Analysis viewpoint is not visible, please restart your Eclipse using **File › Restart**.

Navigate to a Petri Net model element to open an existing graphical view or right click it to create a new one.



A graphical view is opened.



The Petri Net shows how the trace lines are projected upon it.

Black Bar - it represents a trace point transition. A green arrow on top indicates this trace point transition has been passed.

Gray Bar - it represents a transition doesn't require a trace point check.

Red Bar - it represents an armed transition with a certain trace point is expected. Usually, it is a place where an error occur.

Yellow Circle - it represents an action as a place.

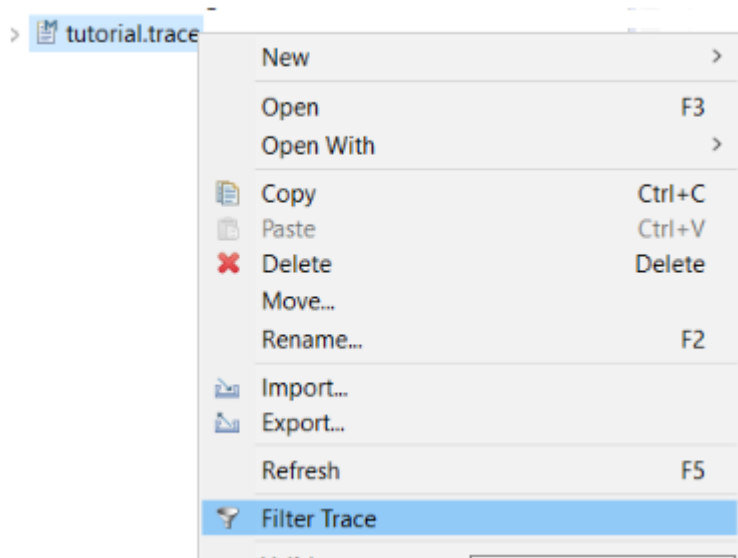
Gray Circle - it is a place to connect two transitions, if there is no action in between.

Blue Token - it indicates where the conformance check stops.

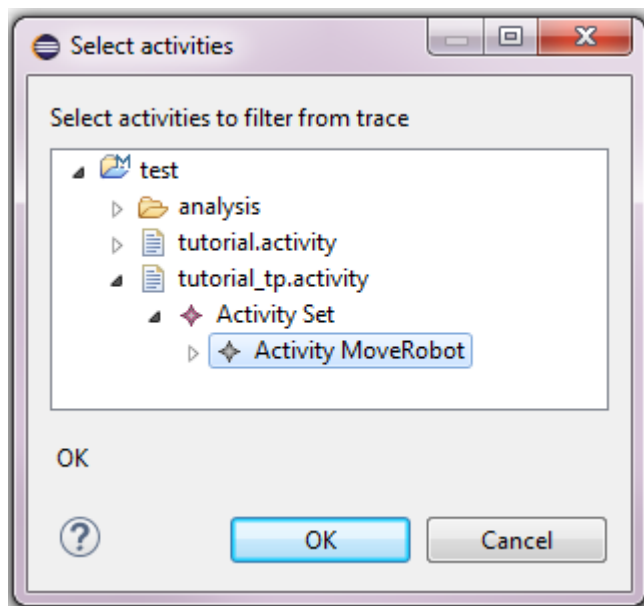
8.3. Filter Trace Point

Filtering out irrelevant trace points in log files is not required for conformance check. However, it helps to reconstruct activities manually, especially when the amount of irrelevant trace points is much more than the relevant ones. Before filtering, activity specification should be inserted with trace points.

Right click log files and select *Filter traces*.



Select *Activity* elements which contain all the relevant trace points and continue with *OK*.



After filtering, selected log files will only keep the trace points which are used in activities.

Chapter 9. Generate Max-Plus Specification

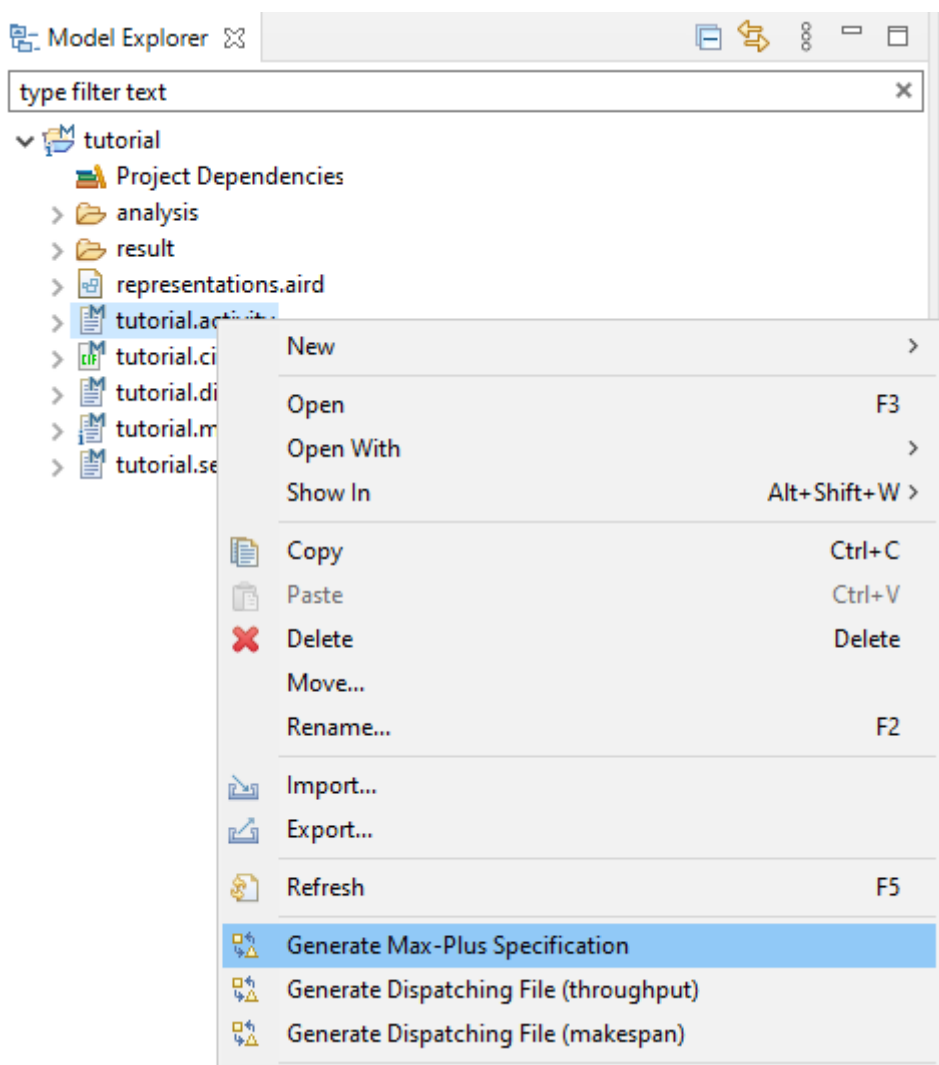
The timing behavior of an activity execution is characterized by two elements, which are:

- synchronization** waiting for incoming claims, releases, or actions to finish
- delay** action duration (specified either directly in the settings file, or derived from a motion specification)

A max-plus matrix can be used to concisely capture this timing behavior. The matrix describes the longest path between each resource claim and release. Computing the total execution time of an activity sequence can be done by multiplying the corresponding max-plus matrices.

The max-plus specification also contains the supervisory controller described by CIF. Given all possible activity sequences and the corresponding max-plus matrices, the worst-case and best-case throughput and makespan can be computed.

To generate a max-plus specification, click on the activity file and select "Generate Max-Plus Specification".



Next, select the respective CIF file. If the CIF file contains cycles, the minimum/maximum throughput can be computed. If the CIF file does not contain cycles, the minimum/maximum

throughput can be computed.

Modify the CIF file as it is shown below in order to include a cycle.

```
controllable MoveRobot; ①

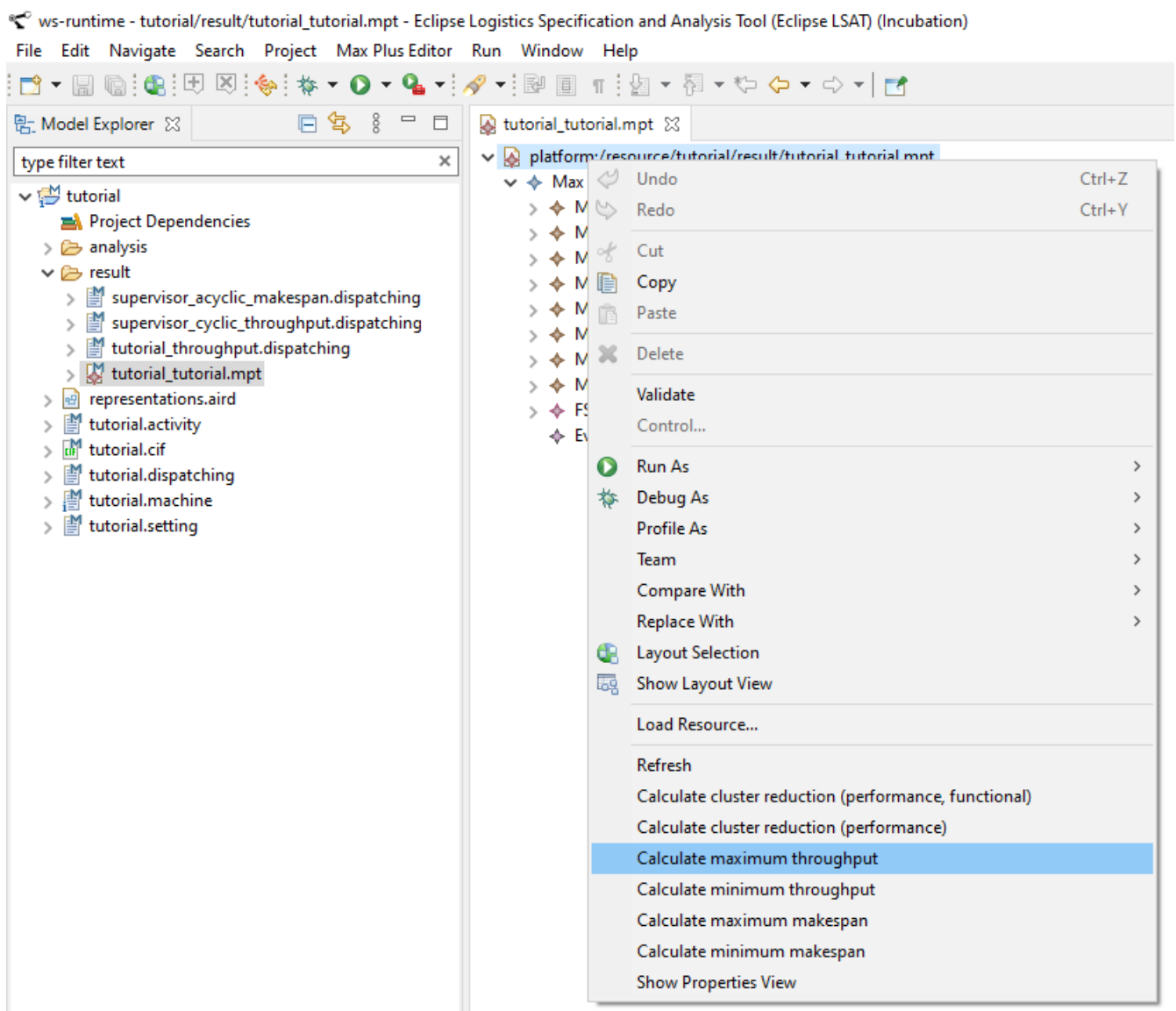
supervisor Tutorial: ②

  location l0:
    initial; ③
    edge MoveRobot goto l1; ④

  location l1:
    edge MoveRobot goto l0;

end
```

Now, we are ready to generate the activity dispatching sequence ensuring the maximum throughput.



The result of the computation is shown in the Error Log, including some additional details about the computation times of the steps.

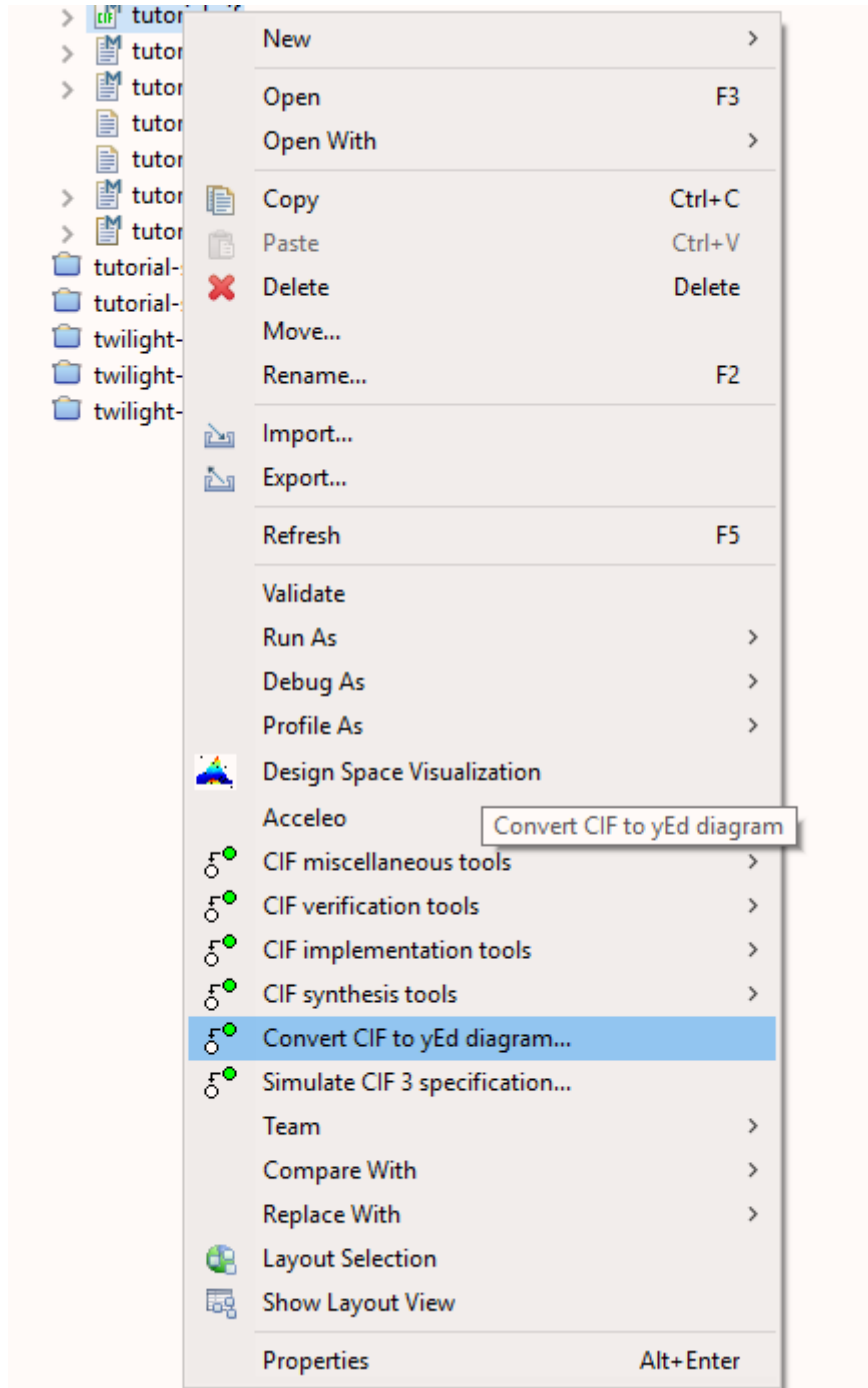
Workspace Log		
Message		Plug-in
 Max plus result: MaximumThroughputResult [throughput=0.17232317488217405, steady state activities=[MoveRobot, MoveRobot], transient state activities=[]]		org.eclipse.lsat.common.mpt.dsl.editor
 Maximum throughput computed using Howards's MCR algorithm in 2 ms.		org.eclipse.lsat.product.branding
 Running Howard on component 1		org.eclipse.lsat.product.branding
 Found 1 strongly connected component(s) in 0 ms.		org.eclipse.lsat.product.branding
 Max-plus state space constructed: 2 states and 2 edges. Generation took 0 ms.		org.eclipse.lsat.product.branding
 Resource Robot2 is not updated by any of the activities. Resource is not taken into account during performance analysis.		org.eclipse.lsat.product.branding
 Resource Robot2 does not have any incoming dependency from another resource. This might lead to generation of an infinite max-plus state space.		org.eclipse.lsat.product.branding



The max-plus specification is used mostly for experimental features, including for example partial-order reduction. Computing the maximum throughput or minimum makespan can also done without first computing the max-plus specification, as described in [Generating an Optimal Activity Dispatching Sequence](#).

Chapter 10. Generate GraphML from CIF specification

Eclipse LSAT™ also gives you the opportunity to generate a GraphML diagram from a CIF specification. This feature allows the user to verifying the correctness of the CIF specification.



To do so, execute the following steps:

1. Right click on the desired file containing the CIF specification.
2. Select *Convert CIF to yEd diagram..* as shown in the figure above.
3. Select your desired output from the available radio buttons: error, warning, normal or debug.
4. Press *OK*.

5. Two new files should be generated:
 - a. (name of your cif file).model.graphml
 - b. (name of your cif file).relations.graphml

For the official documentation on the CIF to yEd transformer, click [here](#).

Chapter 11. Application Programming Interface (API)

Eclipse LSAT™ is equipped with a REST API which can be used to e.g., start a timing analysis. The available REST calls are described in the section [REST calls](#) below. The results of a timing analysis are returned in a JSON format as described [below](#). Note that the same results can also be exported as a JSON file by manually starting a timing analysis via the GUI (see [Scheduling & Configurations](#)).

The Eclipse LSAT™ API data format is specified using OpenAPI. From this specifications clients for various programming languages are generated which can be used to parse the JSON file:

- Java
- Python
- C++

The Java and Python clients will automatically resolve references in the JSON file. For example, instead of writing `task_id = task_ids[0]; task = lsat_data.tasks[task_id]` it is possible to write `task = tasks[0]`. For other clients, like the C++ client, this shorthand notation is not available.

The OpenAPI specification and the clients can be found in the *api-clients* folder of the LSAT installation. Other clients can be generated from the OpenAPI specifications with tools like [OpenAPI Generator](#). An example usage for the Python client is given in [Python API client](#).

11.1. REST calls

The Eclipse LSAT™ API can be accessed via HTTP on port 4560. The following REST calls are available:

- `/api/health`: check if the API is available
- `/api/list`: list all projects present in LSAT
- `/api/motioncalculator`: make a request to the currently configured motion calculator. The single argument is a URL-encoded JSON string which must follow the JSON schema for motion calculator requests as described in the [design documentation](#) of Eclipse LSAT™. A minimal example to query the supported profiles by the motion calculator is as follows:
`http://localhost:4560/api/motioncalculator?request=%7B%22requestType%22%3A%20%22SupportedProfiles%22%7D`
- `/api/reload`: reload a specific project. The name of the project should be provided via the mandatory argument *project*.
- `/api/reloadall`: reload all projects
- `/api/timinganalysis`: start a timing analysis on one of the projects in LSAT

Mandatory arguments for running a timing analysis:

- `project`: name of the project e.g., bowling

- setting: name of the settings file (without extension), e.g., bowling
- dispatch: name of the dispatching file (without extension), e.g., bowlingbowling_alt1

Optional arguments for running a timing analysis:

- goal: select the timing analysis goal, can be *normal*, *critical_path* or *stochastic_impact* (default: *normal*)
- debug: run timing analysis in debug mode (default: *false*)
- sampleLength: number of samples to use when running a stochastic analysis (default: 100)
- colorErroneousMoves: when there are erroneous moves, color them in the Gantt chart (default: *false*)
- strictProductTracing: enforce strict tracing of products, will return a list of unprocessed product changes if the product flow is incomplete (default: *false*)
- exportMotionPlots: enable motion plots, can be *true* or *false* (default: *false*)
- motionPlotSampleFrequency: sample frequency for the motion plots (default: 1000)
- motionPlotFilter: selection of peripherals for which motion plots will be exported (default: empty, which implies motion plots for all peripherals will be exported)
- **/api/validate**: run model validation on the selected project

Mandatory argument for validation:

- project: name of the project e.g., bowling

Optional arguments for validation:

- errors: report validation errors (default: *true*)
- warnings: report validation warnings (default: *true*)
- format: return validation in either *JSON* or *TEXT* (case-insensitive, default: *JSON*)

A minimal example REST call for running the timing analysis on the bowling ball example is as follows: http://localhost:4560/api/timinganalysis?project=bowling&setting=bowling&dispatch=bowling_alt1

To export motion plots for the XY peripheral of the *LoadRobot* and *UnloadRobot*, the REST call can be extended like this: http://localhost:4560/api/timinganalysis?project=bowling&setting=bowling&dispatch=bowling_alt2&exportMotionPlots=true&motionPlotSampleFrequency=1000&motionPlotFilter=LoadRobot.XY%20UnloadRobot.XY

11.2. JSON format

The API's timing analysis JSON response contains both the model used in the timing analysis as well as the results. The model follows the specification as described in [Logistics Specification](#).

The JSON response starts with a list of all peripheral types, followed by all resources and peripherals. Next, variable declarations and their corresponding values are given. Activities contain a list of (references to) actions, which can be looked-up in the actions section. There are various types of actions, such as move or peripheral actions, claim or release actions, or events. Different

type of actions have different properties, such as the start and end position and the movement profile for a move, or the resource and peripheral for a peripheral-action. Also dispatch groups contain a list of (references to) dispatches, which can be looked-up in the dispatches section. Each dispatch has a unique number, a (reference to) its activity and a set of (user defined) attributes.

The result of the timing analysis is split in three parts: all the tasks which are scheduled, the (dependency) graph of these tasks, and the final schedule. All tasks have a unique number and properties like their name, start time, duration, which action and which dispatch they belong to, etc. In case motion plots are enabled in the timing analysis this data is also exported for move-tasks. The graph section contains all dependencies between claim-and-release tasks. Finally the sequence section contains a list of all sequences (i.e., the swim lanes in the Gantt chart) with the corresponding resources and peripherals, and a list of all tasks scheduled in this sequence. The sequence section also lists all dependencies between all types of tasks.

Finally the product tracing results are provided. This includes the life cycle of each projects, i.e., when it enters and exits the system, and when it is transferred between peripherals, and all the updates of the properties of the products.

11.3. Python API client

The Python API client can be used to parse results generated by the timing analysis, either directly from a REST call or from a file. After parsing it is possible to e.g., iterate over all tasks in the resulting schedule, as is shown below:

```
from lsat_client import LsatData ①
import requests

if USE_REST_CALL:
    r = requests.get(
        'http://localhost:4560/api/timinganalysis?project=bowling&setting=bowling&dispatch=bowling_alt1') ②
    json_str = r.text
else:
    with open('bowling_alt1.json', 'r') as f: ③
        json_str = f.read()

lsat_data = LsatData.from_json(json_str) ④

for seq in lsat_data.schedule.sequences: ⑤
    if seq.peripheral is not None:
        print(f'{seq.resource.name}.{seq.peripheral.name}')
    else:
        print(f'Event {seq.resource.name}')
    for task in seq.tasks: ⑥
        print(' - ', task.name)
```

① Import the LSAT API client.

② Either perform a REST call to start a timing analysis and collect the results.

- ③ Or read the results from the file `bowling_alt1.json` which is generated by the timing analysis of the bowling example.
- ④ Parse the contents of the file to an `LsatData` object.
- ⑤ Iterate over all sequences in the schedule and print the corresponding resource and peripheral name.
- ⑥ Iterate over all tasks in a sequence and print the task's name.

The Python client requires the Python packages *pydantic*, *python-dateutil*, and *urllib3*. The Python package *requests* can be used to perform REST calls.

11.4. Java API client

The Java API client can be used to parse JSON files generated by the timing analysis, similar to the Python API client. An example function to convert the JSON data into an `LsatData` object is shown below:

```
import org.eclipse.lsat.external.api.model.LsatData; ①
import org.eclipse.lsat.external.api.util.LsatDataUtil;

public LsatData convert(String json) {
    return LsatDataUtil.fromJson(json); ②
}
```

- ① Import the `LsatData` data class and the `LsatDataUtil` utility class.
- ② Parse a JSON string to an `LsatData` object.

The Java client has the following dependencies: *Jakarta Annotations* and *Gson*.

11.5. C++ API client

The C++ API client can be used to parse JSON files generated by the timing analysis, similar to the Java and Python API client. However, the C++ client does not automatically resolve references in the JSON file.

The C++ client requires [JSON for Modern C++](#), a JSON parser in a single header file, and (optionally) [restclient-cpp](#) for performing REST requests.

An example C++ application to either perform a REST call to start a timing analysis, or to process results from a JSON file, and print the start and end time of the first task in the first sequence is given below:

```
#include <fstream>
#include <iostream>
#include <nlohmann/json.hpp>
#include <restclient-cpp/restclient.h>
```

```

#include "LsatData.h"

int main(int argc, char* argv[])
{
#ifdef USE_REST_CALL
    auto response = RestClient::get(
        "http://localhost:4560/api/timinganalysis?project=bowling&setting=bowling&dispatch=bowling_alt1");
    nlohmann::json data = nlohmann::json::parse(response.body);
#else
    std::ifstream jsonFile(argv[1]);
    nlohmann::json data = nlohmann::json::parse(jsonFile);
#endif
    lsat::model::LsatData lsatData;
    from_json(data, lsatData);

    auto tasks = lsatData.getTasks();
    auto schedule = lsatData.getSchedule();
    auto sequences = schedule.getSequences();
    auto taskIds = sequences[0].getTaskIds();
    auto taskId = taskIds[0];
    auto task = tasks[taskId];

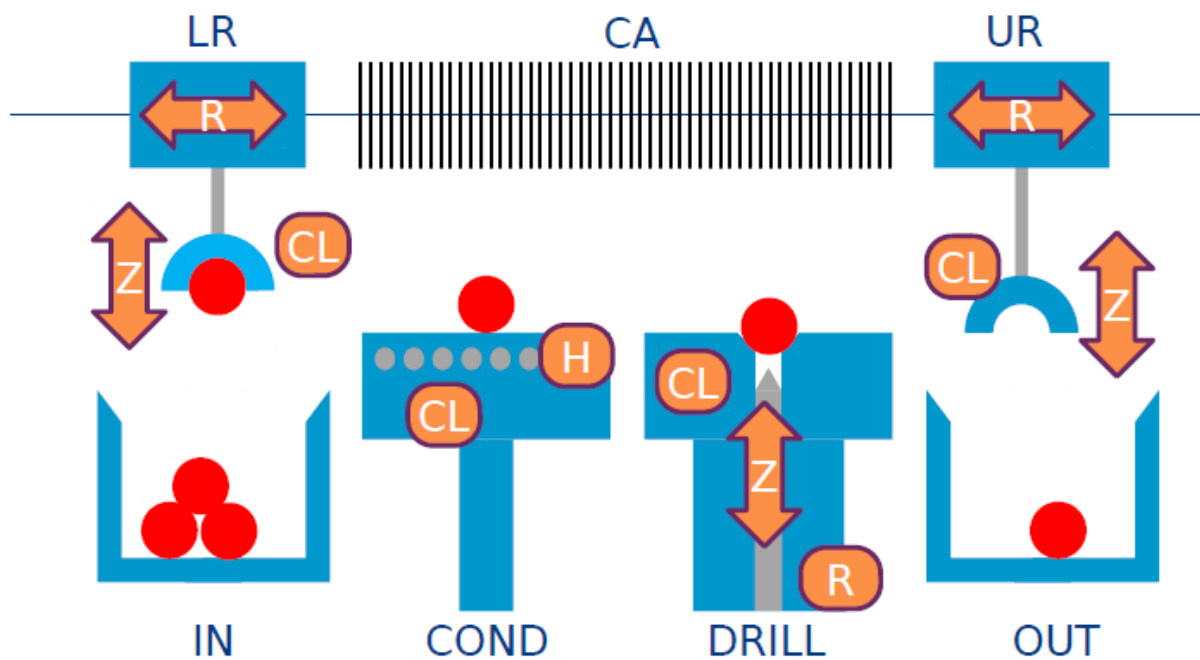
    std::cout << "Task " << task.getName()
        << " starts at " << task.getStartTime()
        << " and ends at " << task.getEndTime()
        << std::endl;
    return 0;
}

```

Chapter 12. Twilight Manufacturing System: a showcase

The first step for using Eclipse LSAT™ in industry is to decompose the manufacturing system into a plant by defining resources, peripherals, actions and the respective activities ordering.

In this section, we introduce the Twilight System to show a use case of Eclipse LSAT™ in industry. Twilight refers to a simplification version of a wafer handling sub-system used at ASML. This system is presented in the figure:



The **components** of a system are described by resources. Each resource consists of one or many peripherals.

The **behavior** of the system is modeled on three levels:

1. Actions: executed by peripherals.
2. Activities: used to describe scenarios of end-to-end deterministic behavior. An activity consists of a set of actions and dependencies among these actions.
3. Activity sequences: used to describe orderings of activities.

Within the respective system, the Resources are Collision Area (CA), Load Robot (LR), Conditioner (COND). Each resource contains usually more than one peripherals, for example resource LR is composed of three peripherals: the R and Z motors and the CL clamp. The motors allow LR to move within a certain area, whereas the CL clamp allows it to grasp/release a product.

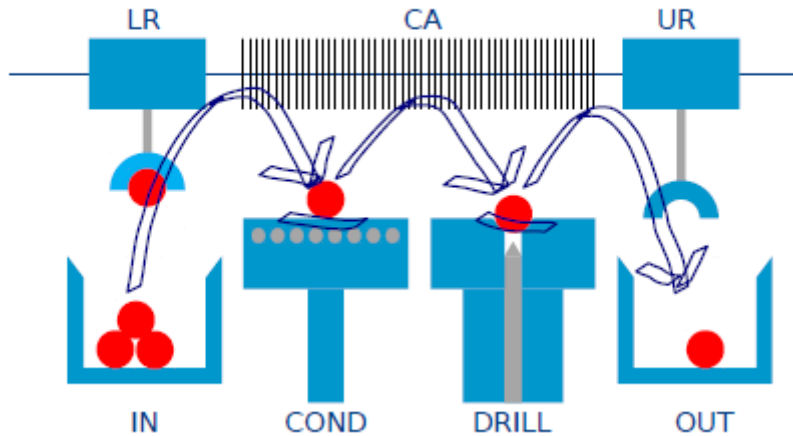
The actions are:

1. Claim and Release resources;

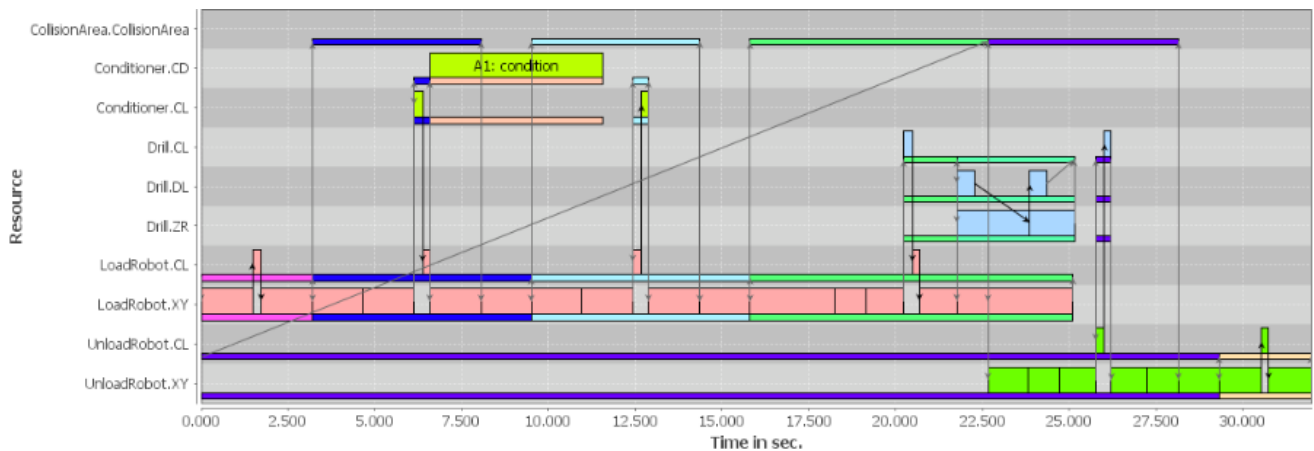
2. Unclamp, Clamp, Moves.

The basic activities are: Condition and Drill, whereas those related to the moving are: LR_PickFromInput, LR_PickFromCond, LR_PickFromDrill, LR_PutOnCond, LR_PutOnDrill, UR_PickFromCond, UR_PickFromDrill, UR_PutOnCond, UR_PutOnDrill, UR_PutOnOutput.

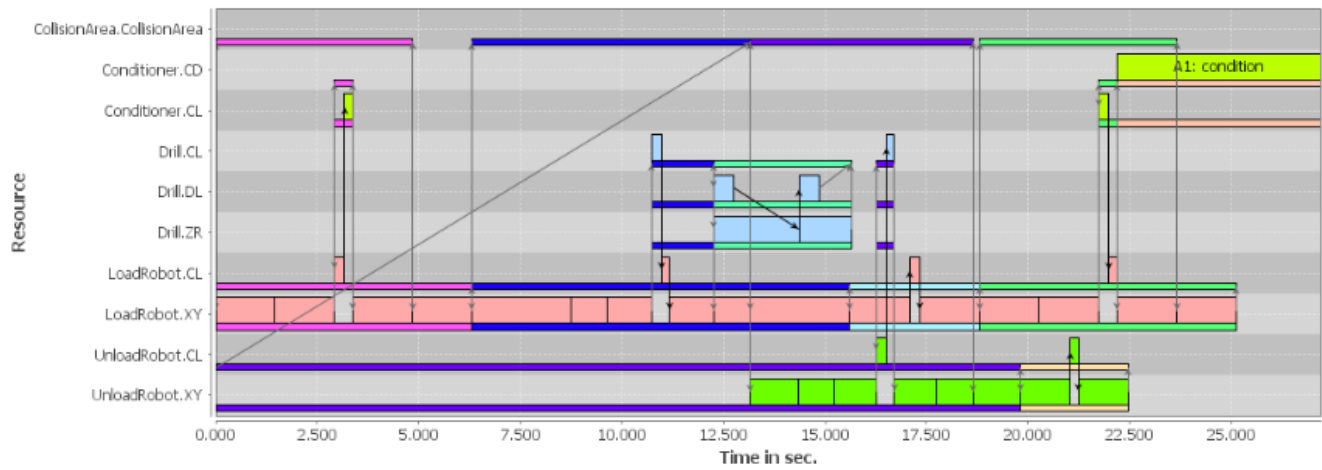
The name of the activities can be easily connected to the respective meaning based on figure:



Based on the dispatching file which specifies the ordering of activities, a feasible schedule can be generated, which meets the constraints related to the resources and the synchronization/delays.



As explained in [Generating an Optimal Activity Dispatching Sequence](#), this schedule can be optimized in terms of throughput/makespan based on the information included in the CIF file.



The figure above shows the ordering of the activities for an optimal throughput.

For an extensive explanation regarding the theory and the examples based on Eclipse LSAT™, please refer to:

1. [Modular Specification and Design Exploration for Flexible Manufacturing Systems, PhD, J. Bastos](#)
2. [Performance Analysis and Optimization of Supervisory Controllers, PhD, B.Sanden](#)

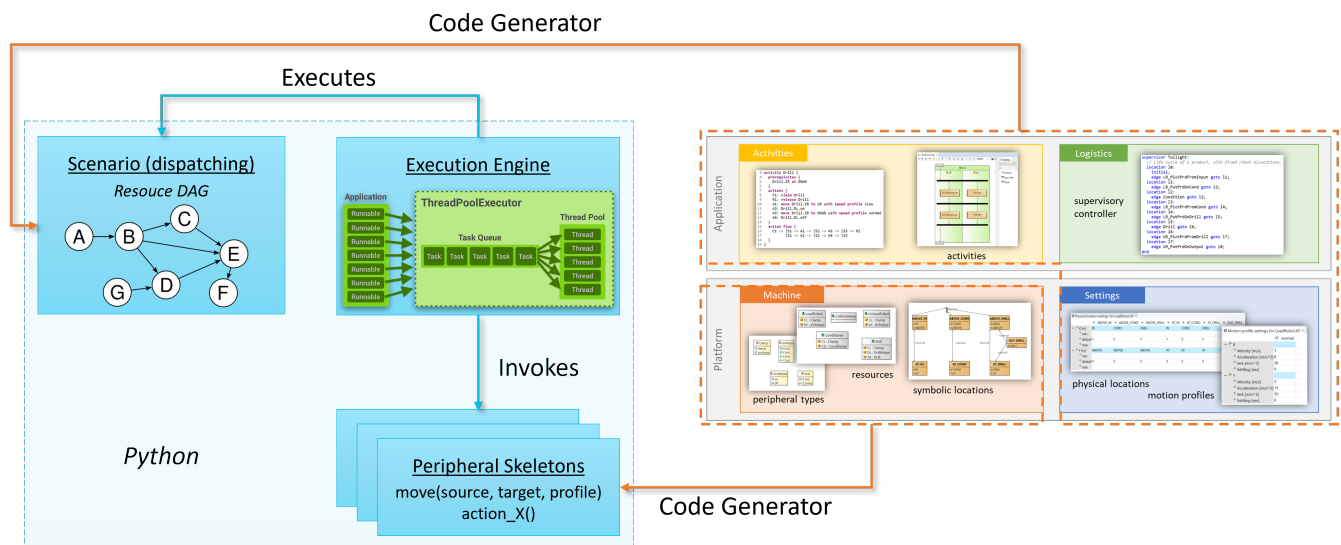
Chapter 13. Python Execution Engine

13.1. Introduction

The execution engine can be used to execute an LSAT scenario (as defined in an LSAT dispatching file) against a physical or virtualized environment. From the LSAT specification a resource task graph is generated which represents tasks, their allocations to resources (peripherals) and their dependencies. The execution engine drives this graph and orchestrates invocation of peripheral behaviour on the target environment.

The engine is implemented using a thread-pool executor where one thread is reserved for every peripheral declared in the specification. Tasks in the resource task graph are allocated to a peripheral and are executed on that peripheral's reserved thread. When the engine executes a task it invokes the corresponding method on the peripheral skeleton.

Peripheral skeletons are generated from the LSAT specification: for each peripheral a Python skeleton class is created that provides placeholder methods for all simple actions and move operations the peripheral can perform. Integrators should implement or extend these skeleton classes so the placeholder methods perform the actual interactions with physical or virtual devices.



Generated peripheral skeleton example

The LSAT tooling generates a Python skeleton class for each peripheral declared in the specification. Below is a short example (based on the Tutorial) showing the style of the generated action and move methods. Integrators should add their implementation between the "Start of user code" and "End of user code" comment markers.

```
#!/usr/bin/env python

...

def robot1_m_apply_brake(**kwargs):
    """
    Executes action apply_brake on peripheral Robot1.M
```

```

"""
getTracer('Robot1.M').info('"Robot1" > "apply_brake"')
getLogger('Robot1.M').info('> robot1_m_apply_brake')
# Start of user code robot1_m_apply_brake
# End of user code
getLogger('Robot1.M').info('< robot1_m_apply_brake')
getTracer('Robot1.M').info('"Robot1" < "apply_brake"')

def robot1_m_move(**kwargs):
    """
    Moves peripheral Robot1.M to the specified position.
    By default this is called from the specialized move methods below
    :param **kwargs['dest']: specifies the target position of the move
    """

    getLogger('Robot1.M').info('> robot1_m_move (source=\'%s\', dest=\'%s\')', kwargs
.get('source'), kwargs.get('dest'))
    # Start of user code robot1_m_move
    # End of user code
    getLogger('Robot1.M').info('< robot1_m_move')

```



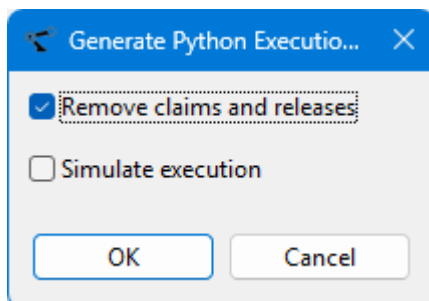
Place your implementation code between the "Start of user code" and "End of user code" comments so the generator can safely regenerate the surrounding skeleton without losing your custom logic.

13.2. Usage

Typical workflow

1. Generate the execution artifacts from a dispatching model

Generate the execution engine artifacts directly from a `.dispatching` file: right-click the `.dispatching` file in the Project Explorer (or the appropriate navigator), then select the context menu item menu:[Generate Python Execution Engine...]. This opens the Execution Engine generation dialog (see the image below). The generator produces the Python peripheral skeletons and a runner that wires the resource task graph to the execution engine.



2. Implement peripheral behaviour

Edit the generated peripheral skeletons and add the platform-specific code between the "Start of user code" and "End of user code" markers in each method. This preserves your changes when the skeletons are regenerated.

3. Run the scenario

For each `.dispatching` file a Python runner script with the same name is generated in the `src-gen` folder of your project. This runner file executes the scenario and will drive the resource task graph and call into your implemented peripheral methods.

- a. **From the command line:** run the generated Python runner directly with your configured Python interpreter:

```
python src-gen/mydispatching.py
```

- b. **From within Eclipse:** optionally install the PyDev plugin for Eclipse to enable direct execution of Python files from the IDE. With PyDev installed, right-click the generated runner file and select **Run As > Python Run**.
4. Monitor execution and logs The engine emits tracing and logging messages (the generated code uses `getTracer` and `getLogger` calls). Inspect the Eclipse Console, the LSAT execution log and the Error Log view for progress, errors and Python tracebacks.

Notes

- The Python Execution Engine is experimental. Expect that generated code or the engine internals may change between releases; always preserve user code in the marked regions and keep the generator and engine versions aligned.
- For advanced deployments (remote devices, containers or bespoke device APIs) implement the peripheral skeleton methods to delegate to the appropriate communication layer or driver.

13.3. Installation

The Python Execution Engine is provided as an optional Eclipse feature: the "LSAT Python Execution Engine Feature (Experimental)". It can be installed from the Eclipse LSAT P2 (update site) repository .

13.3.1. Install from the P2 repository (Eclipse GUI)

1. In Eclipse select **Help > Install New Software...**
2. Click **[Add...]**, give the update site a name (for example "LSAT Updates") and enter the Eclipse LSAT™ P2 repository URL used by your installation. Please consult the Eclipse LSAT™ website for the exact update-site URL.
3. From the "Work with" drop-down select the LSAT update site. Expand the categories and locate "LSAT Python Execution Engine Feature (Experimental)".
4. Select the feature, click **[Next]**, accept any licenses and complete the install. Restart Eclipse when prompted.

13.3.2. Notes and verification

- The Python Execution Engine feature is optional and marked experimental in the LSAT product. It depends on core Eclipse LSAT™ features (for example `org.eclipse.lsat.scheduler.feature`); the p2 repository should resolve these automatically when installing.

- After installation verify the feature is present via **Help › About Eclipse IDE › Installation Details › Installed Software** and look for the label "LSAT Python Execution Engine Feature (Experimental)".
- You can also check the installed bundles list for `org.eclipse.lsat.engine.python` and `org.eclipse.lsat.engine.python.ui`.

Chapter 14. Release Notes

The release notes of the Eclipse Logistics Specification and Analysis Tool (Eclipse LSAT™) are listed below, in reverse chronological order.

14.1. Eclipse LSAT™ v0.6 (2026-07-31)

This release focuses on:

- Enhancing the REST API with motion calculator and model validation capabilities for programmatic access
- Improving the user experience through visual enhancements to the Gantt chart (colors, fonts, and swimlane reordering)
- Robust product flow tracking and visualization with better product modeling error visibility
- Better error messages across the IDE, including duplicate activities, and path restrictions
- Critical bug fixes in product modeling export, measurement units, and QVTO resource handling in standalone tests

For a complete overview of issues that are part of this release, please see the [milestone v0.6 issue list](#).

The highlights for this release are:

14.1.1. Enhancements

- **REST API for Motion Calculator:** Exposed the motion calculator via the REST API, making powerful motion calculations available for programmatic access and integration with external systems. ([Work Item #196](#))
- **API-Based Model Validation:** Added model validation capabilities to the LSAT REST API, enabling users who work with external editors to receive valuable validation feedback similar to the IDE. ([Work Item #209](#))
- **Synchronous Project Reload via API:** Fixed project reload operations via the API to wait until the project has actually reloaded before returning, providing proper feedback to callers. ([Work Item #170](#))
- **Throughput per Resource Export:** Added export functionality for the Throughput per Resource Editor data, allowing data to be exported as CSV files for further analysis. ([Work Item #180](#))
- **Gantt Chart Visualization Improvements:** Enhanced Gantt chart usability with:
 - Improved color schemes with better contrast and lighter colors for easier viewing
 - Customizable font sizes to improve readability without requiring zoom
 - Single-axis versus multi-axis visualization for motion plots with toggleable options
 - Custom reordering of swimlanes via drag-and-drop for easier analysis ([#185](#), [#186](#), [#190](#), [#194](#))

- **Improved Error Messages:** Enhanced error messaging for better user experience:
 - Cleaner error reporting for duplicate activities with errors tied to the correct location
 - More understandable error message for FullMesh path restrictions
 - Better validation error display in settings DSL outline ([#193](#), [#183](#), [#229](#))
- **Product Flow Visualization:** Added ability to highlight product flow in the Gantt chart, making it easier to visually track how products are processed through the system. ([Work Item #203](#))
- **Enhanced Stochastic Analysis:** Improved product modeling robustness and error visibility:
 - Allow product tracing to show product flow even when there are errors in the model
 - Introduce "Strict product tracing" mode alongside optional debug tracing
 - Better product slot matching based on slot names and activity relationships ([Work Item #191](#))
- **Distance Parameter Support:** Added support for Distance as a parameter type in parameterized activities. ([Work Item #223](#))
- **Toggable Multiple Axes Warning:** Made the warning for moving multiple axes simultaneously toggleable, as it is not necessary for all users. ([Work Item #224](#))
- **Math functions:** Extended the expressions in the settings DSL with the power operator (^) and math functions like **Round**, **Floor**, and **Ceil** for more flexible calculations. ([Work Item #233](#))
- **PERT Distribution Default Value:** Changed PERT distribution to use the mean value (instead of mode) as the default for standard timing analysis, providing consistency with stochastic analysis. ([Work Item #181](#))
- **Robust Minimum Makespan Analysis:** Enhanced the minimum makespan analysis to handle CIF models with cycles, allowing for more realistic system models. ([Work Item #202](#))
- **SBOM Generation in Build:** Added Software Bill of Materials (SBOM) generation to the Eclipse LSAT release build process as preparation for compliance with the EU Cyber Resilience Act. ([Work Item #176](#))
- **QVTO Transformation:** QVTO transformations have been replaced by Java implementations. ([Work Item #163](#))

14.1.2. New Documentation

- **Stochastic Distributions User Guide:** Added comprehensive documentation for working with stochastic distributions in LSAT. ([Work Item #105](#))
- **Python Execution Engine Documentation:** Created user documentation covering installation, configuration, usage, and result interpretation for the Python execution engine. ([Work Item #166](#))

14.1.3. Bug Fixes

- **Incorrect Measurement Units:** Fixed measurement unit display to consistently show all measurements in seconds and productivity computed as units per hour. ([Work Item #179](#))
- **Stochastic Analysis Product Export:** Fixed incomplete product life cycle in exported JSON files

when both "Export to JSON" and "Enable product tracing" are enabled during stochastic analysis. ([Work Item #178](#))

- **Throughput per Resource with Events:** Fixed error when opening the Throughput per Resource Editor with models containing events (e.g., Bowling Ball example). ([Work Item #182](#))
- **Settings DSL Auto-complete:** Fixed auto-complete functionality to work correctly when adding Resource.Peripheral references at the start of setting files. ([Work Item #184](#))
- **Text Editor Keywords:** Made VAJ, units, and SCAN keywords appear bold and be properly recognized as keywords in text editors. ([Work Item #187](#))
- **Incomplete Product Modeling Information in Export:** Added product modeling information (slot names, in/out relationships) to the JSON export even when product life cycles are incomplete, aiding in debugging. ([Work Item #199](#))
- **Inconsistent Position Export:** Fixed export of source and target positions in actions to correctly reference position definitions instead of location dictionaries. ([Work Item #198](#))
- **CIF Analysis Compatibility:** Fixed CIF analysis for models using parameterized activities (e.g., updated Bowling Ball example). ([Work Item #207](#))

14.1.4. Architecture/Code Improvements

- **Timing Analysis Refactoring:** Separated timing analysis execution from UI code in the `TimingAnalysisLaunchDelegate` class, improving code organization and enabling timing analysis to be started from external APIs. ([Work Item #146](#))

Software upgrades

- Eclipse TRACE4CPS v0.3 → TRACE4CPS v0.4

14.2. Eclipse LSAT™ v0.5 (2026-02-06)

This release contains bugfixes, improvements and new features. For a complete overview of issues that are part of this release, please see the [milestone v0.5 issue list](#).

The highlights for this release are:

14.3. Enhancements

- **Motion Plots for Robot Moves:** Added visualization of speed, acceleration, jerk, and setpoint locations in addition to axis locations, providing better insight into robot movements over time. ([Issue #74](#))
- **ALAP/Just-in-Time/Deadlines Support:** Extended LSAT with support for modeling and analyzing actions and activities that should execute as-late-as-possible or meet specific deadlines. ([Issue #81](#))
- **Stochastic Distributions in Expression Language:** Added support for assigning stochastic distributions to variables in settings files, providing consistency with direct timing value assignments. ([Issue #94](#))
- **Additional Statistical Distributions:** Added Gamma, LogNormal, and Poisson distributions to

the available options in settings. ([Issue #97](#))

- **Motion Calculator Post-Processing:** Implemented a system for applying post-processing operations to deterministic motion calculator results, allowing for stochastic behavior in calculated timings through the new **MoveAdjustments** section. ([Issue #121](#))
- **Fixed Duration Profile:** Added support for fixed duration moves in the P2P Motion Calculator, simplifying the specification of moves with predetermined durations. ([Issue #124](#))
- **Math functions:** Extended the expressions in the settings DSL with the modulo operator and math functions like **sin**, **cos**, **tan**, **asin**, **acos**, **atan**, **atan2**, **abs**, **exp**, **log**, **min**, **max** and **sqrt** for more flexible calculations. ([Issue 115](#) and [Issue 126](#))
- **Enhanced Stochastic Impact Analysis:** Added comprehensive statistics to stochastic impact analysis, including mean, standard deviation, skewness, min, and max values for action start times. ([Issue #132](#))
- **HTTP/REST Server:** Added a REST API server to retrieve data and execute functions on LSAT, supporting project listing, file management, and timing analysis operations. ([Issue #133](#))
- **Parameterized Activities:** Introduced parameter support for activities, similar to arguments in functional programming languages, allowing for more reusable activity definitions and clearer specifications. ([Issue #139](#))
- **Chart View CSV Export:** Added functionality to export Chart View data (including distance, velocity, acceleration, and jerk over time) as CSV files for better analysis. ([Issue #140](#))
- **Product Modeling System:** Implemented a comprehensive system to track product ownership transfers and property changes via the generated schedule. Products can be owned and transferred between peripherals, with all transfers and modifications recorded in JSON activity files after timing analysis.
- **Socket/REST/IP based SPG:** Added support for socket-based JSON communication for custom Setpoint Generators, making it easier to integrate with other languages like MATLAB and Python without the complexity of shared C-libraries. ([Issue #110](#))
- **Machine Diagram Behavior:** Improved user experience by making machine diagrams open only on user request rather than automatically. ([Issue #137](#))
- **Performance Improvements:** Enhanced performance for large models, particularly in stochastic analysis, event removal, and activity validation. ([Issue #138](#))

14.4. Bug Fixes

- **Documentation Fix:** Corrected the description in CONTRIBUTING.asciidoc which incorrectly contained TRACE description instead of LSAT description. ([Issue #72](#))
- **QVTO Distribution Recognition:** Fixed an issue where QVTO distributions were not being recognized in Eclipse editors and Project explorer, despite working correctly in Eclipse and Maven builds. ([Issue #103](#))
- **Chart View Limits Display:** Resolved incorrect plotting of limits in the chart view when displaying concatenated moves with different movement profiles. ([Issue #117](#))
- **Stochastic Impact Analysis:** Fixed calculation errors in stochastic impact analysis that were causing incorrect critical path identification due to rounding discrepancies. ([Issue #131](#))

- **Duplicate Peripheral Types:** Addressed an issue where machine files allowed multiple peripheral types with the same name, causing only the first defined peripheral type's actions to be recognized. ([Issue #136](#))
- **Scientific Notation Support:** Fixed parsing of numbers in scientific notation, particularly for values like 1.0e2 which were not being recognized correctly in settings and time-constraint limits. ([Issue #150](#))

14.5. Eclipse LSAT™ v0.4 (2025-08-08)

This release contains bugfixes, improvements and new features. For a complete overview of issues that are part of this release, please see the [milestone v0.4 issue list](#). The highlights for this release are:

- **Embedded Java runtime**

"It is no longer necessary to install a Java Runtime Environment (JRE) for LSAT, as [JustJ](#) is now embedded within LSAT

Software upgrades

- Eclipse 2020-06 → Eclipse 2024-09
- Eclipse ESCET v0.6 → [Eclipse ESCET v6.0](#)
- Eclipse TRACE4CPS v0.1 → [Eclipse TRACE4CPS v0.3](#)
- Eclipse JustJ → [Eclipse JustJ 21](#)

14.6. Eclipse LSAT™ v0.3 (2024-02-15)

This release contains bugfixes, improvements and new features. For a complete overview of issues that are part of this release, please see the [milestone v0.3 issue list](#). The highlights for this release are:

- **Parameterized events**

To avoid duplicate activities, Eclipse LSAT™ already allowed pools of resources and parametrized activities. Parameterization is added for events to avoid activity duplication if events are used for synchronization.

Software upgrades

- Eclipse TRACE4CPS v0.1 → [Eclipse TRACE4CPS v0.2](#)

14.7. Eclipse LSAT™ v0.2 (2023-02-07)

This release contains bugfixes, improvements and new features. For a complete overview of issues that are part of this release, please see the [milestone v0.2 issue list](#). The highlights for this release are:

- Add support for passive claims in LSAT (issue [#38](#))

A resource can be passively claimed to ensure that no actions are performed by any of its peripherals until the resource is released again. If more activities claim a resource passively,

they potentially can run in parallel.

14.8. Eclipse LSAT™ v0.1 (2022-11-01)

This release contains the Eclipse LSAT™ initial contribution. This version is a port of [ESI LSAT v2.0.1](#) that is prepared for its release to the Eclipse foundation, including:

Software upgrades

- Eclipse Neon → Eclipse 2020-06
- [ESI TRACE v0.63](#) → [Eclipse TRACE4CPS v0.1](#)
- [TU/e CIF3](#) → [Eclipse ESCET v0.1](#)

Modularity/Flexibility

- Modular setting files
- Expression language
- Resource pools; more concise specifications
- User-defined attributes in dispatching file e.g., for product tracking
- Multiple phases in dispatching file e.g., set-up, steady-flow and tear-down
- Dependencies between activities

Motion calculator

- Motion profiles: support for rotational movements
- APIs for integration of motion libraries (JSON and/or Java)