



Elements of System Dynamics

Antoine B. Rauzy

PART 3. THE WORLDLAB™ WORKSHOP

LECTURE 8. INTERFACING AND DOCUMENTING

LECTURE 8. INTERFACING AND DOCUMENTING

Key notions:

- Language WIDL
- Language TAU

Learning Objectives

This lecture aims at presenting two languages (other than Σ^{TM}) that are used in the WorldLabTM Workshop:

WIDL is a simple but powerful language to specify graphical user interfaces of interactive simulations of Σ^{TM} models.

TAU is a mark-up language, similar to LaTeX, that is used to document models, experiments and results of experiments.

Tau documents can be compiled into:

- LaTeX documents which can in turn be compiled into PDF files.
- HTML pages.

The HTML pages produced by the Tau compiler can be embedded in the WorldLabTM Workshop projects.

Agenda

Chapter I. WIDL

Section 1. The WIDL Language

Section 2. Generic Views

Section 3. Diagrams

Section 4. Expressions and Object-Oriented Constructs

Agenda

Chapter II. TAU

Section 5. The TAU Language

Section 6. Front-End Material

Section 7. Sectioning and Formatting

Section 8. Figures and Tables

Section 9. References and Hyperlinks

LECTURE 8. INTERFACING AND DOCUMENTING

SECTION 1. THE WIDL LANGUAGE

Rational

Interactive simulations make it possible for the analyst to go forth and back in the execution of Σ^{TM} models and to **visualize** at each step the values of variables, ports, observers...

Interactive simulations pursue actually two objectives:

- **Debugging** models.
- **Sharing** the models and **discussing** with **stakeholders**.

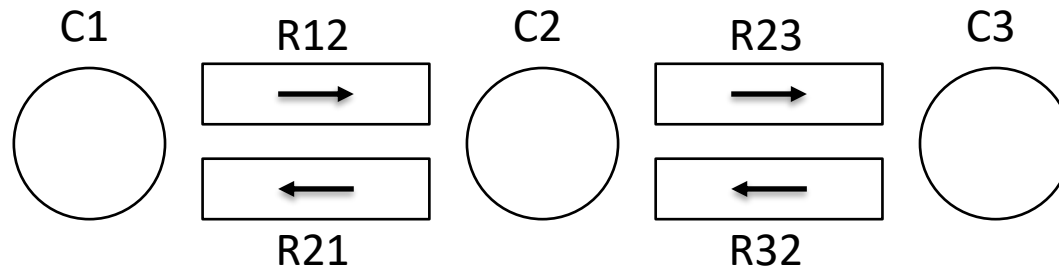
To make interactive simulations user friendly and attractive, it is often of interest to design dedicated simulation interfaces, typically obeying the client visual identity guidelines.

WIDL* is a simple but powerful object-oriented language to specify graphical user interfaces of interactive simulations of Σ^{TM} models.

(*) WIDL stands for (WorldLab Interface Description Language)

Case Study

Throughout this presentation, we shall consider the following road network.



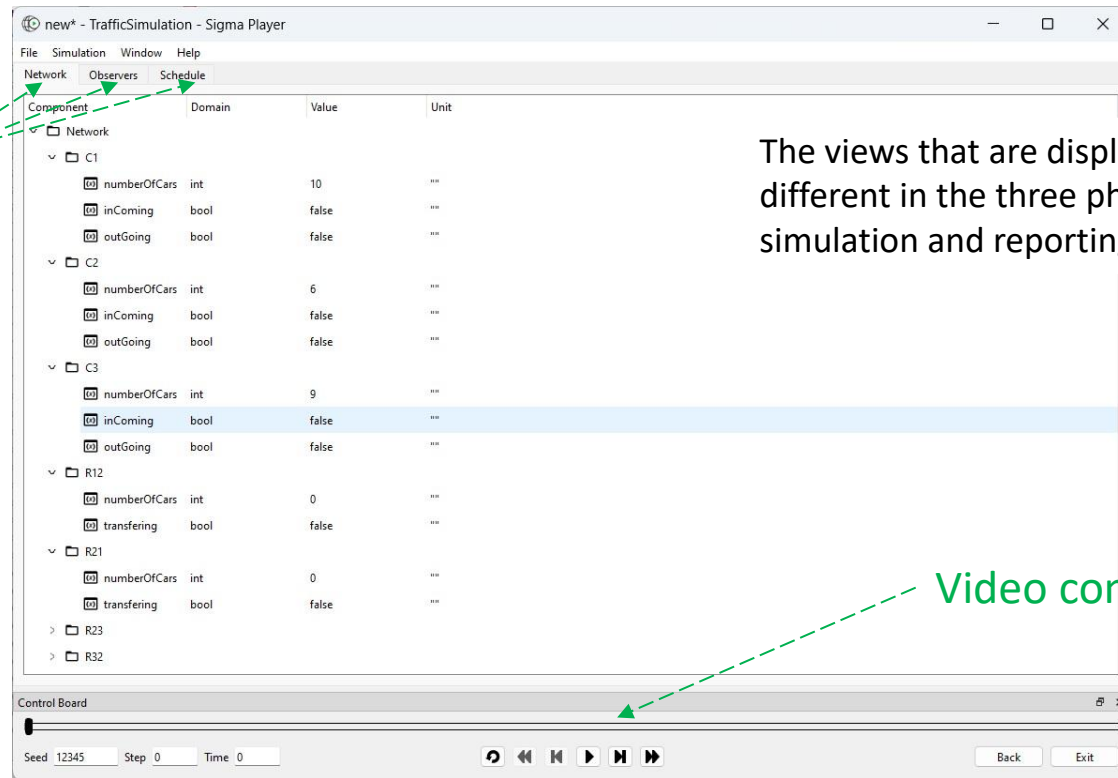
The network consists of three crossings C1, C2 and C3, and four roads R12, R21, R23 and R32 joining these crossings.

Cars go in and out and circulate in this network.

The Σ^{TM} Player

Interactive simulations are performed using the Σ^{TM} **player**, which works according to the **video player metaphor**. Several **views** on the state of the system and the simulation are available.

Views

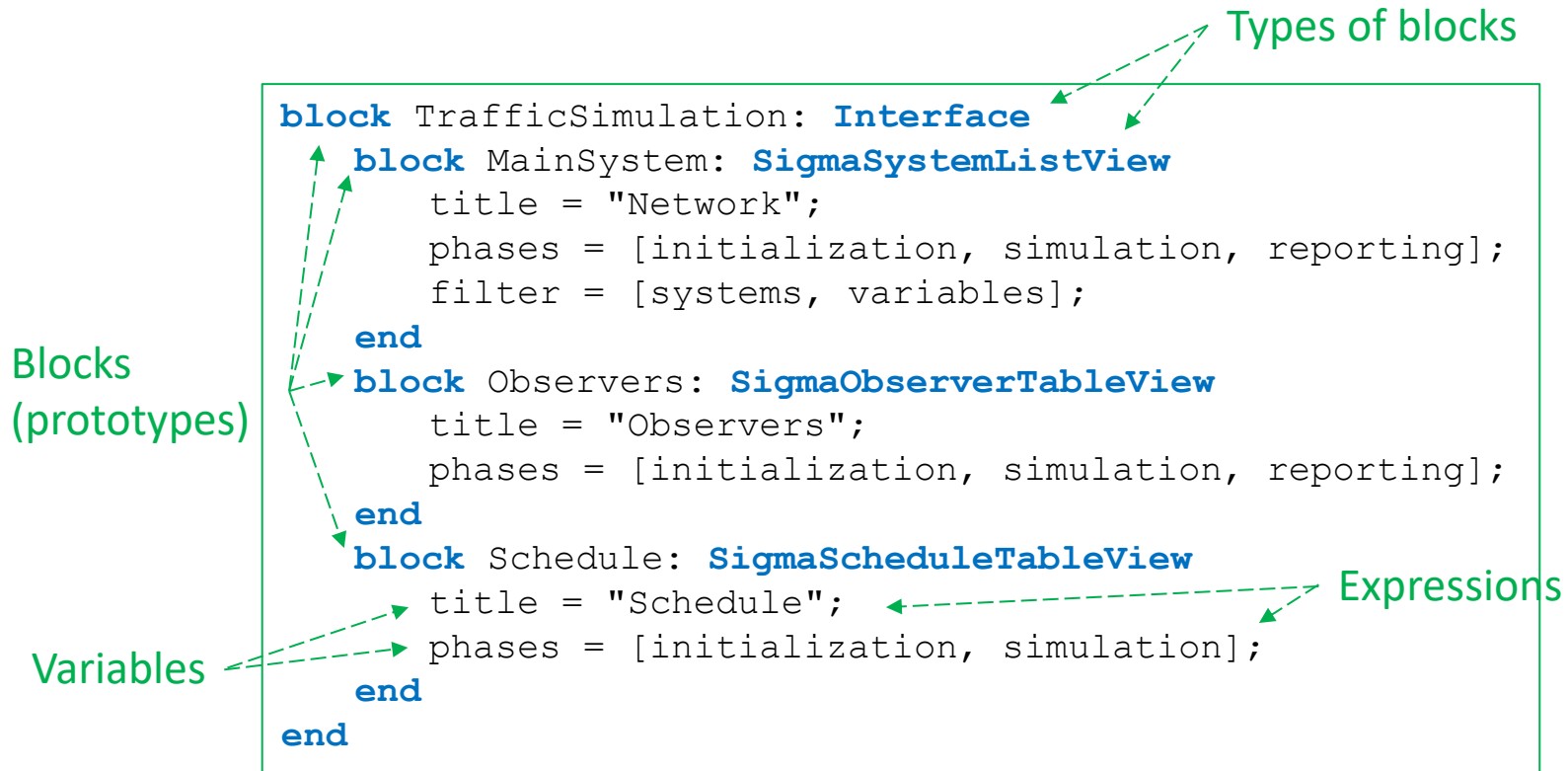


The views that are displayed may be different in the three phases: initialization, simulation and reporting.

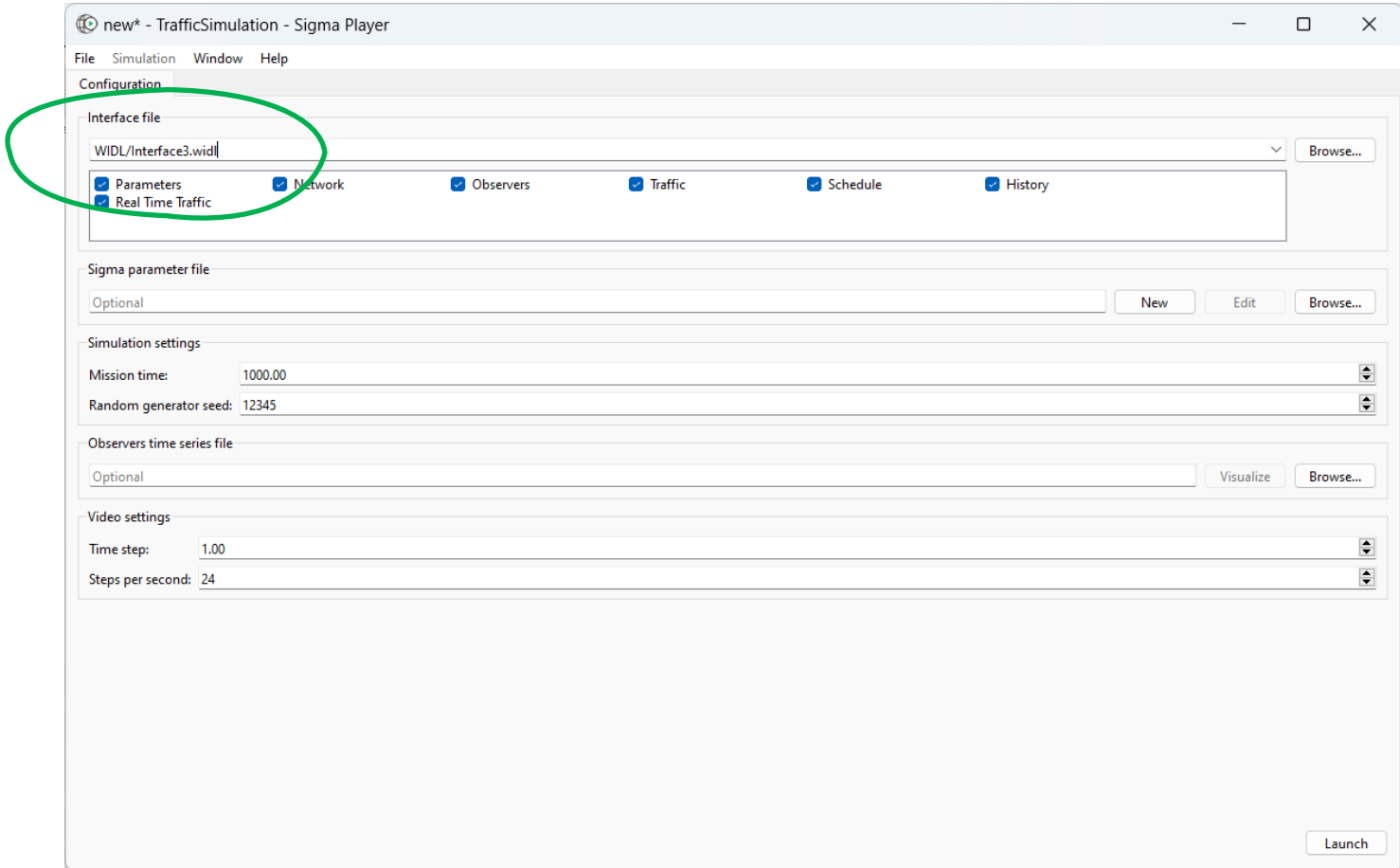
Video control board

WIDL

WIDL is thus a full-fledged object-oriented, domain specific language dedicated to the description of the views to be embedded into the Σ^{TM} Player.



Loading WIDL File



LECTURE 8. INTERFACING AND DOCUMENTING

SECTION 2. GENERIC VIEWS

Parameters and Systems

	Parameter	Domain	Value	Unit
1	Network.C1.initialNumberOfCars	int	10	""
2	Network.C1.inComingRate	float	0.05	"s ⁻¹ "
3	Network.C1.outGoingRate	float	0.05	"s ⁻¹ "
4	Network.C2.initialNumberOfCars	int	6	""
5	Network.C2.inComingRate	float	0.05	"s ⁻¹ "
6	Network.C2.outGoingRate	float	0.05	"s ⁻¹ "
7	Network.C3.initialNumberOfCars	int	9	""
8	Network.C3.inComingRate	float	0.05	"s ⁻¹ "
9	Network.C3.outGoingRate	float	0.05	"s ⁻¹ "
10	Network.R12.meanTransferTime	float	12	"s"
11	Network.R21.meanTransferTime	float	11	"s"
12	Network.R23.meanTransferTime	float	10	"s"
13	Network.R32.meanTransferTime	float	11	"s"

```

block Parameters: SigmaParameterTableView
  title = "Parameters";
  phases = [initialization];
end

```

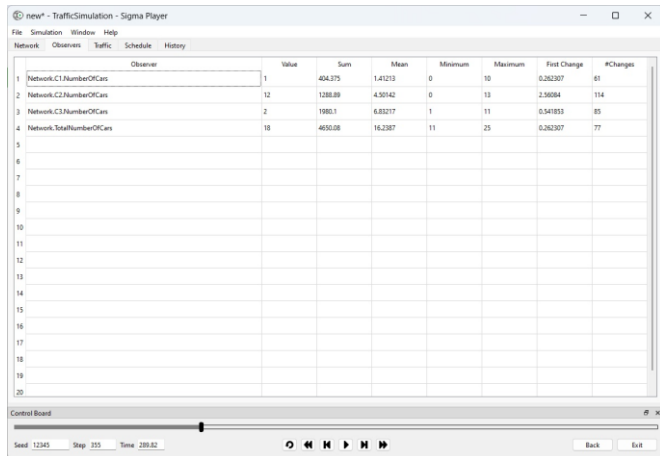
Component	Domain	Value	Unit
Network			
C1			
numberOfCars	int	1	""
inComing	bool	true	""
outGoing	bool	true	""
C2			
numberOfCars	int	12	""
inComing	bool	true	""
outGoing	bool	true	""
C3			
numberOfCars	int	2	""
inComing	bool	true	""
outGoing	bool	true	""
R12			
numberOfCars	int	1	""
transferring	bool	true	""
R21			
numberOfCars	int	1	""
transferring	bool	true	""
R23			
numberOfCars	int	1	""
transferring	bool	true	""
R32			
numberOfCars	int	1	""
transferring	bool	true	""

```

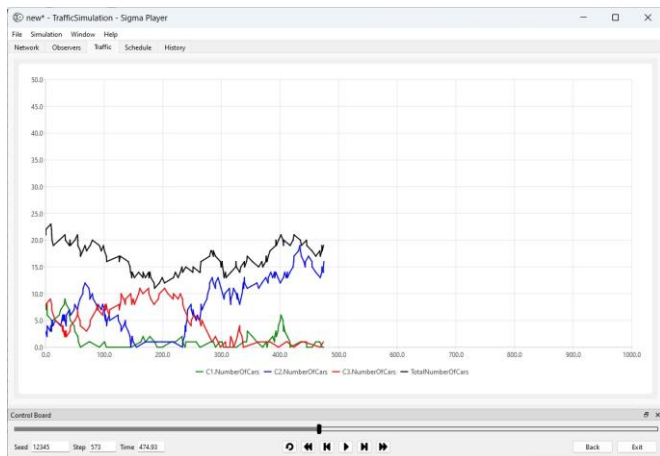
block Network: SigmaSystemListView
  title = "Network";
  phases = [initialization, simulation,
            reporting];
  filter = [systems, parameters, variables,
            observers];
end

```

Observers and Time Series



```
block Observers: SigmaObserverTableView
    title = "Observers";
    phases = [simulation, reporting];
end
```



```
block TimeSeries: SigmaTimeSeriesView
    title = "Traffic";
    yMax = 50;
    xTicks = 11;
    yTicks = 11;
    block TotalNumberOfCarsAtC1:
        SigmaTimeSeries
        observer = "C1.NumberOfCars";
        color = "green";
        interpolation = linear;
    end
    ...
    legend = true;
    legendPosition = "bottom";
    phases = [simulation, reporting];
end
```

Schedule and History

Step	Time	Network	Actor	Activity	Event
1	575	470.036	Network.C2	outGoingCar	completion
2	575	470.957	Network.C3	outGoingCar	completion
3	576	476.155	Network.C3	inComingCar	completion
4	577	481.9	Network.B2	transfer	completion
5	578	485.458	Network.B2	transfer	completion
6	578	485.113	Network.B2	transfer	completion
7	580	491.045	Network.C1	inComingCar	completion
8	581	496.928	Network.C2	inComingCar	completion
9	582	513.477	Network.C1	outGoingCar	completion
10	583	1000			observation
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

```
block Schedule: SigmaScheduleTableView
  title = "Schedule";
  phases = [initialization, simulation,
            reporting];
end
```

Step	Time	Network	Actor	Activity	Event
1	572	476.93	Network.B12	transfer	completion
2	572	476.297	Network.B12	transfer	start
3	571	476.297	Network.C1	inComingCar	start
4	576	476.297	Network.C1	inComingCar	completion
5	566	474.32	Network.B12	transfer	completion
6	566	473.912	Network.B12	transfer	start
7	567	473.912	Network.B23	transfer	start
8	566	473.912	Network.B23	transfer	completion
9	565	472.016	Network.C1	outGoingCar	start
10	564	472.016	Network.C1	outGoingCar	completion
11	563	471.426	Network.C2	inComingCar	start
12	562	471.426	Network.C2	inComingCar	completion
13	561	471.237	Network.C2	inComingCar	start
14	560	471.237	Network.C2	inComingCar	completion
15	559	466.702	Network.B12	transfer	start
16	558	466.702	Network.C3	outGoingCar	start
17	557	466.702	Network.B13	transfer	start
18	556	466.702	Network.B13	transfer	completion
19	555	467.085	Network.C1	inComingCar	start
20	554	467.085	Network.C1	inComingCar	completion

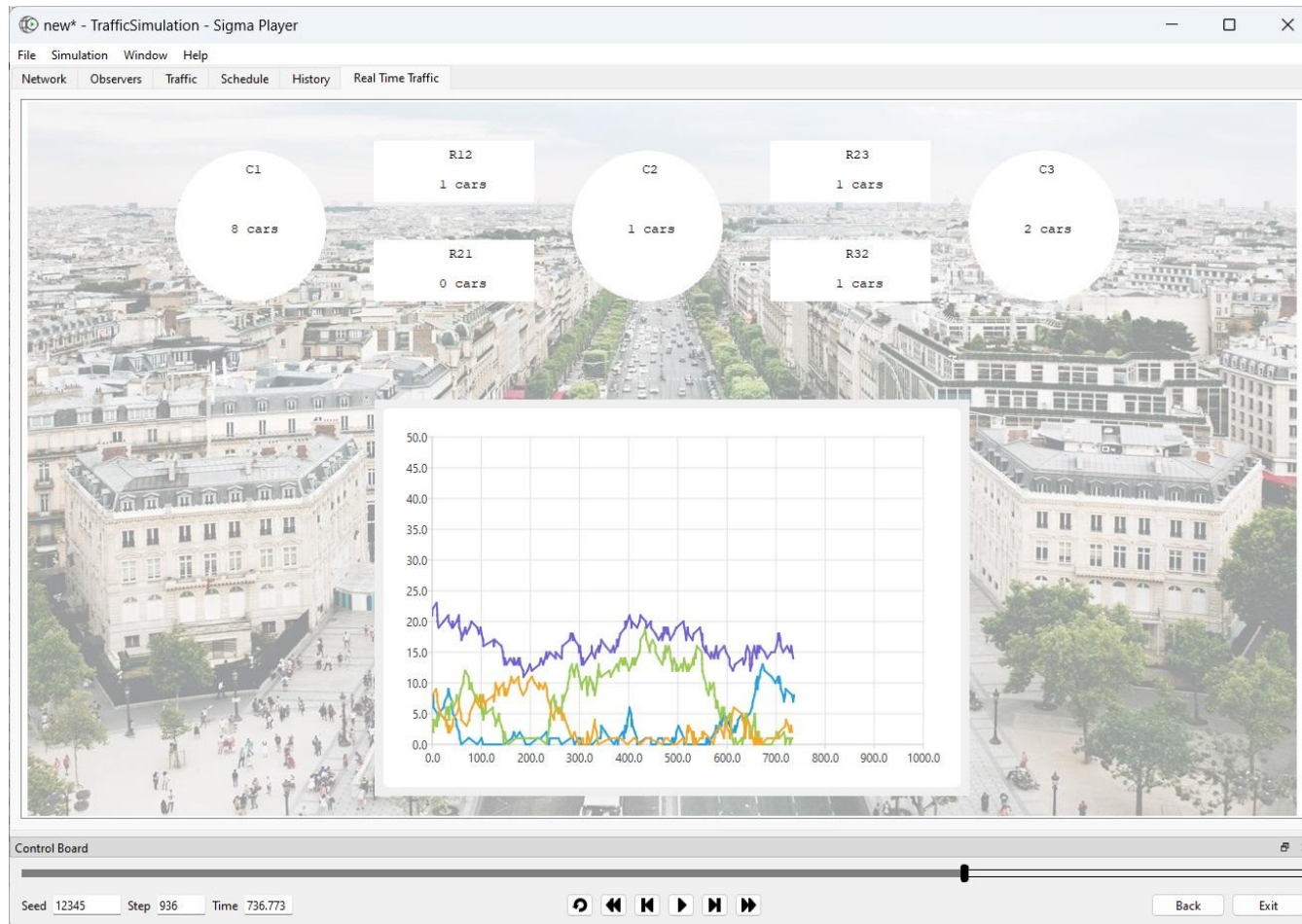
```
block History: SigmaHistoryTableView
  title = "History";
  phases = [simulation];
  length = 20;
end
```

LECTURE 8. INTERFACING AND DOCUMENTING

SECTION 3. DIAGRAMS

Diagrams

Diagrams implement *ad-hoc* views. A diagram is a **scene** containing **rectangles**, **ellipses**, **lines**, **texts** and **images**. More structured objects are also available.



Diagrams

```
block TrafficSimulation.TrafficDiagram: Diagram
  title = "Traffic";
  phases = [simulation, reporting];
  backgroundImage = "WIDL/Champs Elysees Light.jpg";
  scale = 1.0;
  block TrafficText: Text
    x = 50;
    y = 50;
    text = string{{Network.C1.numberOfCars}} + " cars";
  end
end
```

Position

Reference to variables and observers of the model

Basic Graphical Objects

block RoadZone: **Rectangle**

```
x = 140;  
y = 10;  
width = 200;  
height = 100;  
foregroundWidth = 2;  
foregroundColor = "black";  
backgroundColor = "white";  
toolTip = "Current area";
```

end

block CrossingZone: **Ellipse**

```
x = 140;  
y = 10;  
width = 200;  
height = 100;  
foregroundWidth = 2;  
foregroundColor = "black";  
backgroundColor = "white";  
toolTip = "Current area";
```

end

block MyLine: **Line**

```
x1 = 140;  
y1 = 10;  
x2 = 200;  
y2 = 100;  
foregroundWidth = 2;  
foregroundColor = "red";  
foregroundStyle = "dash";  
toolTip = "Occasional link";
```

end

block Icon12: **Image**

```
x = 10;  
y = 25;  
numberOfCars =  
    {{Network.R12.numberOfCars}};  
file = if numberOfCars >= 10  
    then "Images/LowTrafficIcon.jpg"  
    else "Images/HighTrafficIcon.jpg";  
toolTip = "Traffic density";
```

end

Groups and Embedded Views

```
block MyGroup: Group
  x = 10;
  y = 25;
  block SubBlock1 ... end
  block SubBlock2 ... end
  ...
end
```

Positions of child blocks are determined relatively to the position of the parent block.

```
block TimeSeries: SigmaTimeSeriesView
  x = 350;
  y = 300;
  width = 600;
  height = 400;
  title = "Traffic";
  ...
end
```

Generic views can be embedded into diagrams.

LECTURE 8. INTERFACING AND DOCUMENTING

SECTION 4. EXPRESSIONS AND OBJECT-ORIENTED CONSTRUCTS

Expressions

WIDL provides a wide range of expressions (see reference manual):

- Constants;
- Internal and external references;
- Boolean expressions;
- Inequalities;
- Arithmetic expressions;
- Conditional expressions;
- Various builtin functions;
- Lists.

```
numberOfCars = {{Network.R12.numberOfCars}};  
file = if numberOfCars>=10  
      then "Images/LowTrafficIcon.jpg"  
      else "Images/HighTrafficIcon.jpg";
```

Paths

```
block Plant: Diagram
  ...
  block Train1: DiagramBlock
    block Unit1: Unit
      block In: DiagramPort
        ...
      end
    end
  end
  ...
end
block SourceConnector: DiagramConnector
  ...
  block Line1: DiagramConnectorLine
    ...
    target = owner.owner.Train2.Unit1.In;
    ...
  end
  ...
end
...
```

Classes and Instances

```
class Unit: DiagramBlock
    state = {{_state}};
    width = 60;
    height = 70;
    foregroundWidth = 0;
    foregroundColor = "white";
    backgroundColor = "#EEEEEE";
    block UnitIcon: Image ... end
    block Label: Text ... end
    block In: DiagramPort ... end
    block Out: DiagramPort ... end
end
```

```
block Train1: DiagramBlock
    ...
    block Unit1: Unit
        x = 20;
        y = 10;
        Label.x = -10;
        Label.text = "Train1.Unit1";
        state =
            {{Plant.Train1.Unit1._state}};
    end
    block Unit2: Unit ... end
    ...
end
```

Cloning, Inheritance and Import

```
block RedundantLines.PlantDiagram: Diagram
  ...
  block Train1: DiagramBlock ... end
  clones Train1 as Train2;
  Train2.y = 120;
  Train2.Unit1.Label.text = "Train2.Unit1";
  Train2.Unit2.Label.text = "Train2.Unit2";
  Train2.Unit1.state = {{Plant.Train2.Unit1._state}};
  Train2.Unit2.state = {{Plant.Train2.Unit2._state}};
end
```

```
class Pump: BlockDiagram
  extends Unit;
  ...
  block In: DiagramPort ... end
  block Out: DiagramPort ... end
  ...
  Label.text = "Pump";
end
```

```
block TrafficSimulation: Interface
  projectDirectory = "..";
  ...
  block TrafficDiagram: Diagram
    ...
  end
end

import "WIDL/TrafficDiagram.widl";
```

LECTURE 8. INTERFACING AND DOCUMENTING

SECTION 5. THE TAU LANGUAGE

Presentation

TAU is a **markup language**. Primarily designed to write the technical documentation of WorldLab™, it is embedded in the WorldLab™ Workshop to document projects, case studies, experiments ...

Tau documents can be compiled either as **HTML** pages or as **LaTeX** documents, themselves compiled into **PDF** files.

Tau directives are actually very close to those of LaTeX.

```
#+document
#+head
#title Laplace's Demon
#author Marquis Pierre Simon de Laplace
#-head
#+body
#makeFrontEnd
...
#-body
#-document
```

Markups

TAU involves three main types of markups:

Environments are markups for textual elements that spread over multiple lines:

#+abstract

```
#:Tau is a lightweight markup language for creating formatted text  
using a plain-text editor. It is an alternative to the original  
Markdown that lacks both structure and expressive power.
```

#-abstract

Directives are markups for elements that spread over one line, usually titles.

#section Presentation

Tags are markups for elements that are inserted into a larger text.

```
(*it This sentence is italicized.*)  
(*image Figure/EscherWaterfall.jpg*)  
(*cite gruber2013markdown*)
```

Directives

Tau provides directives implementing a wide number of features:

- Front-end material
- Sectioning and formatting
- Lists
- Figures
- Tables
- Tree views
- References and hyperlinks
- Computer code
- Mathematics
- Citations and quotes
- Index

LECTURE 8. INTERFACING AND DOCUMENTING

SECTION 7. LISTS

Lists

Unordered lists

```
#+itemize  
#item List entries start with the header (*code #item *).  
#item Individual entries are indicated with a so-called bullet  
      (a black dot).  
#item The text in the entries may be of any length, contain as many  
      paragraphs as one wants, sub-lists, maths&hellip;.  
#-itemize
```

Ordered lists

```
#+enumerate  
#item Items are numbered automatically.  
#item The numbers start at 1 with each use of the (*code enumerate *)  
      environment.  
#item Another entry in the list  
#-enumerate
```

Description Lists

Description lists

```
#+description  
#item[#:Sigma:] A modeling language of the S2ML+X family.  
#item[#:WIDL:] A language to describe graphical user interface of  
Sigma interactive simulators.  
#item[#:Tau:] A lightweight markup language to write the  
WorldLab documentation.  
#-description
```

Formatting Lists

Style of counters:

```
#+enumerate[numberStyle=roman]  
#item Rectangle,  
#item Ellipse,  
#item Line.  
#-enumerate
```

Start index of ordered lists:

```
#+enumerate[numberStyle=Alphabetic, start=3]  
#item Rectangle,  
#item Ellipse,  
#item Line.  
#-enumerate
```

LECTURE 8. INTERFACING AND DOCUMENTING

SECTION 8. FIGURES AND TABLES

Figures

```
#+figure[align=center]
#+target HTML
(*image[scale=0.5] ../Figures/LogicalArchitecture.jpg*)
#-target
#+target LaTeX
(*image[scale=0.6] ../Figures/LogicalArchitecture.pdf*)
#-target
#caption Escher's Waterfall (270x345 pixels)
#label secFigures:figEscherWaterfall1
#-figure
```

LECTURE 8. INTERFACING AND DOCUMENTING

SECTION 9. REFERENCES

References and Hyperlinks

LECTURE 8. INTERFACING AND DOCUMENTING

SECTION 5. WRAP-UP

Wrap-Up