



Elements of System Dynamics

Antoine B. Rauzy

PART 1. INTRODUCTION

LECTURE 2. GETTING STARTED

LECTURE 2. GETTING STARTED

Key notions:

- Systems, Variables, Activities
- Trigger, Action at Start, Action at Completion, Duration
- Discrete Event Simulations, Interactive Simulations, Stochastic Simulations
- Σ^{TM} , WIDL, and Tau languages
- WorldLabTM Workshop, Σ^{TM} Player, Σ^{TM} Calculator
- Systemic Digital Twins

Learning Objectives

This lecture aims at introducing the **modeling language Σ^{TM}** and **Σ^{TM} ontology** that consists of three fundamental concepts: **systems**, **variables** and **activities**.

It provides also a guided tour of the **integrated modeling and simulation environment WorldLabTM Workshop**.

It discusses the whys and wherefores of:

- **Interactive simulations** of Σ^{TM} models using the **Σ^{TM} Player**.
- **Stochastic simulations** of Σ^{TM} models using the **Σ^{TM} Calculator**.

Finally, it introduces the concept of **systemic digital twin**.

Agenda

Section 1. The Σ^{TM} Ontology

Section 2. The Σ^{TM} language

Section 3. The WorldLabTM Workshop

Section 4. Interactive Simulations (Σ^{TM} Player)

Section 5. Dedicated Simulation Interfaces (WIDL)

Section 6. Stochastic Simulations (Σ^{TM} Calculator)

Section 7. Documentation (Tau)

Section 8. Wrap-Up

LECTURE 2. GETTING STARTED

SECTION 1. THE Σ^{TM} ONTOLOGY

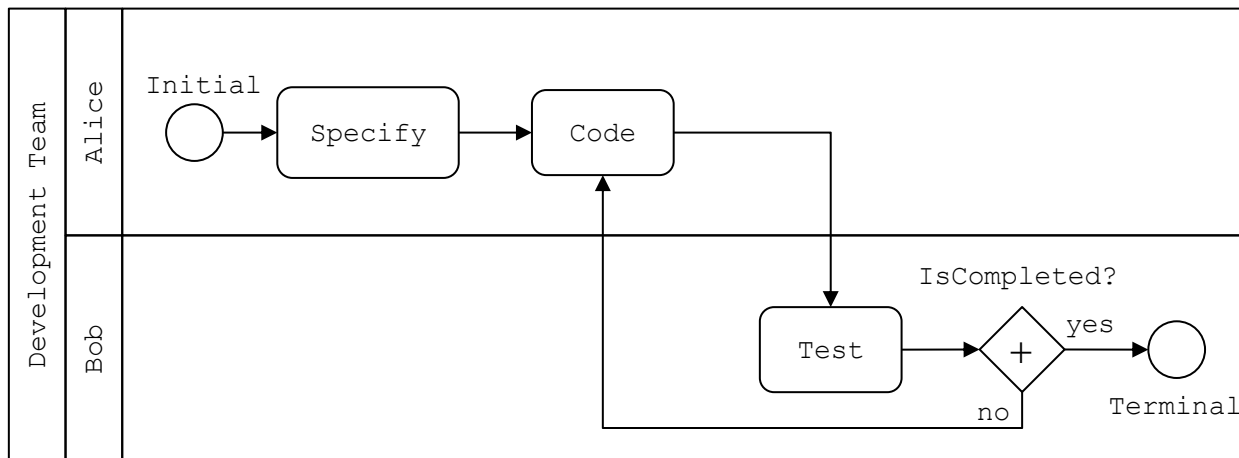
The Σ^{TM} Ontology

The Σ^{TM} **modeling approach** relies on three pillars:

1. A decomposition of the system of interest into a **hierarchy of subsystems**.
2. A characterization of the **state of each subsystem** by means **variables**. Variables takes values into **domains** such as Booleans, integers, reals, sets of symbolic constants...
3. A description of **activities** performed in the system. Activities are the only means by which the system changes of state. They are organized into **processes**.

Case Study: Software Development

Business process for the development of a software

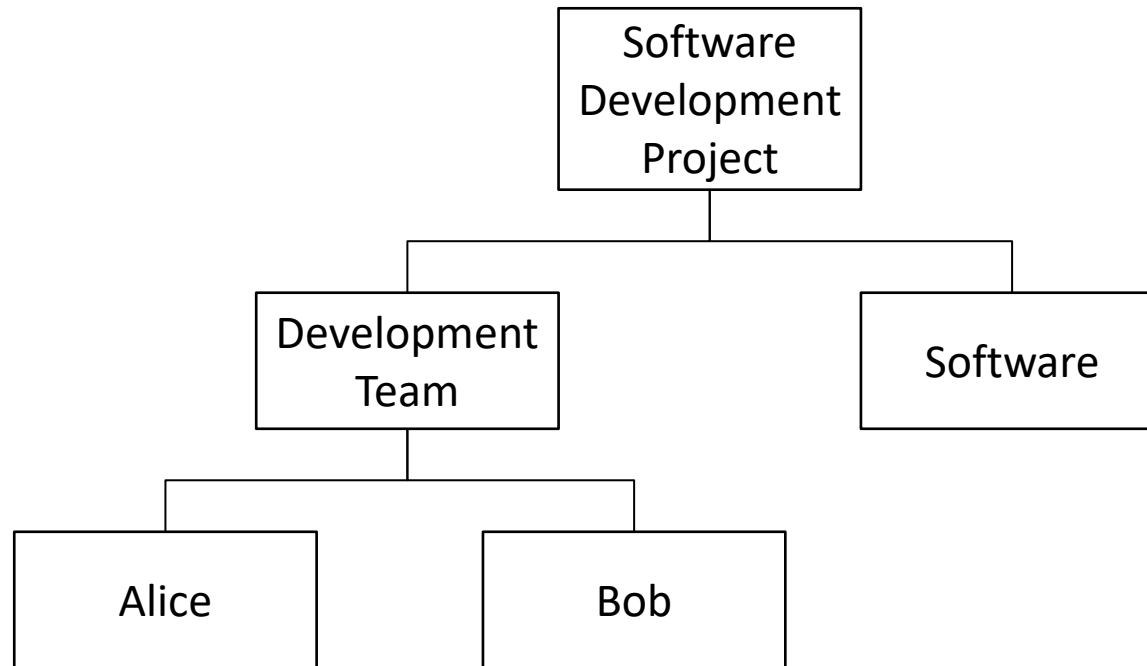


The above diagram obeys the **Business Process Modeling and Notation (BPMN)**.

Recommended reference:

Stephen White and Derek Miers. BPMN Modeling and Reference Guide: Understanding and Using BPMN. Future Strategies Inc.. Lighthouse Point, FL, USA. ISBN 978-0977752720. 2008.

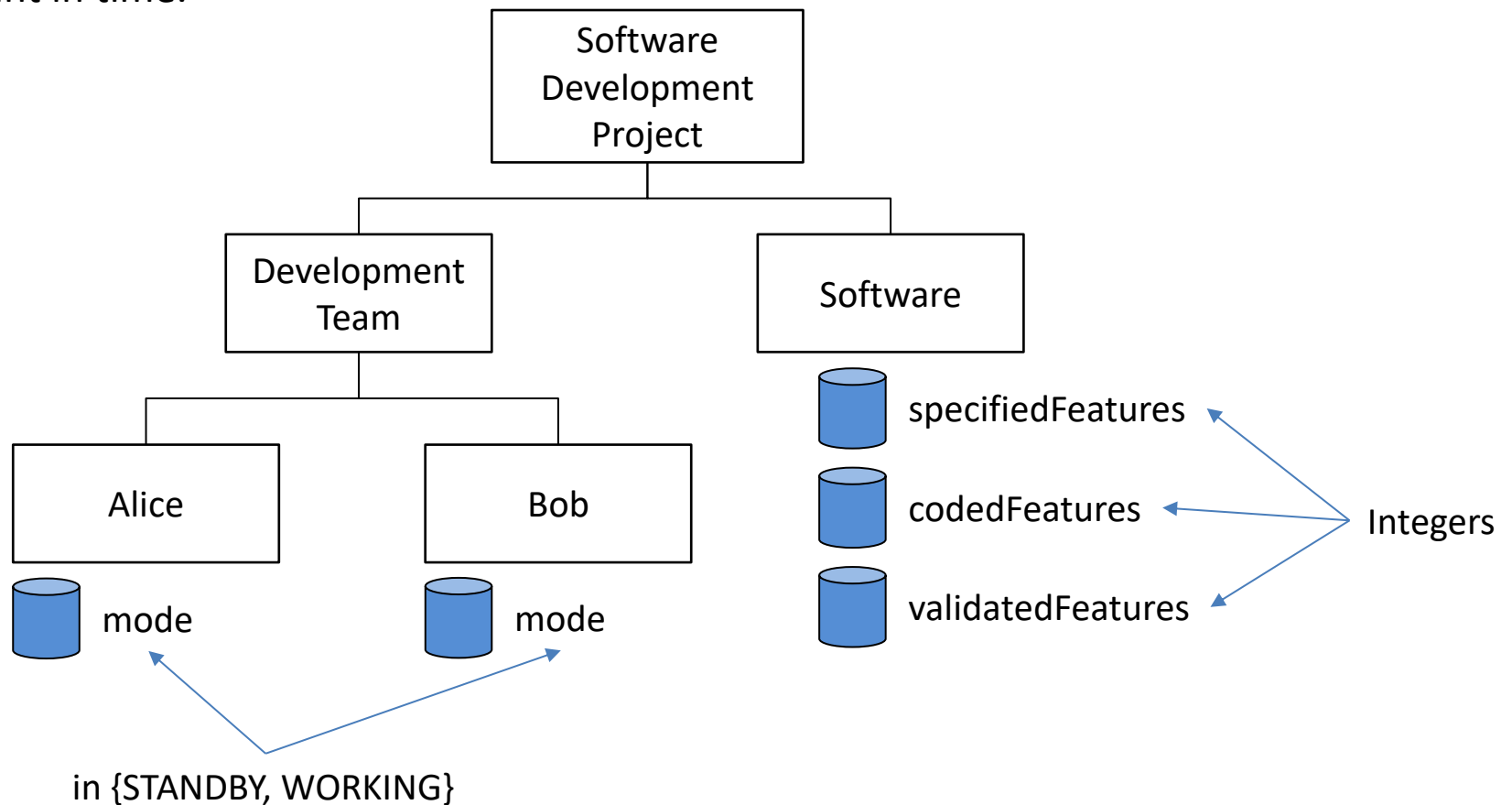
Hierarchical Decomposition



The **hierarchy of (sub)systems** can involve any number of levels and any number of subsystems for each system. The number of levels may differ from branch to branch.

States and Variables

Variables are used to describe the **state** of the system and its subsystems at any point in time.



Activities

Activities are characterized by:

- The condition under which the activity is started, called its **trigger**.
- The resources required by the activity. These resources are booked by means its **action at start**.
- The change on the system performed by the activity described by its **action at completion**.
- Its **duration**.

Software Development

Actor	Activity	Trigger	Action at start	Action at completion	Duration
Alice	Specify	mode = STANDBY specifiedFeatures = 0 validatedFeatures = 0	mode $\leftarrow$ WORKING	specifiedFeatures $\leftarrow$ 12 mode $\leftarrow$ STANDBY	3 weeks
Alice	Code	mode = STANDBY specifiedFeatures $\geq$ 2	mode $\leftarrow$ WORKING specifiedFeatures $\leftarrow$ specifiedFeatures - 2	codedFeatures $\leftarrow$ codedFeatures + 2 mode $\leftarrow$ STANDBY	2 weeks
Bob	Test	mode = STANDBY codedFeatures $\geq$ 4	mode $\leftarrow$ WORKING codedFeatures $\leftarrow$ codedFeatures - 4	mode $\leftarrow$ WORKING validatedFeatures $\leftarrow$ validatedFeatures - 4	3 weeks

We assume here that features are first specified, then coded (and no longer to specify) and eventually tested and validated (and no longer to code).

Collective Exercise

The **life-cycle** of a product is usually made of three phases: **design**, **operation** and **decommissioning**. The operation phase is itself decomposed into two sub-phases: **production** and **maintenance**.

Question 1. Design a BPMN like diagram that represent this life-cycle.

Question 2. Make a Σ^{TM} analysis to implement this diagram assuming that:

- The design phase takes 18 months.
- The system operates 11 months before being maintained for 1 month.
- The system is decommissioned after 10 years.
- Decommissioning the system takes 6 months.

LECTURE 2. GETTING STARTED

SECTION 2. THE Σ^{TM} LANGUAGE

Structure and State of the System

```
domain Mode {STANDBY, WORKING}

system SoftwareDevelopment
  system Company
    system Alice
      Mode mode (init = STANDBY);
    end
    system Bob
      Mode mode (init = STANDBY);
    end
  end
  system Software
    int specifiedFeatures (init = 0);
    int codedFeatures (init = 0);
    int validatedFeatures (init = 0);
  end
end
```

Keywords are in **blue**.

Identifiers are those of programming languages such as Python:

[a-zA-Z_] [a-zA-Z0-9_]*

Lower- and upper-case letters are distinguished.

Attributes **init** set the initial values of variables.

White spaces, tabulations and end-of-lines (any number) are **separators**.

System declarations terminate with the keyword **end**. Variable declarations terminate with a semicolon.

Activity: Specify

```
activity SoftwareDevelopment.Company.Alice.Specify
  trigger:
    return mode==STANDBY and main.Software.specifiedFeatures==0 and
      main.Software.testedFeatures==0;
  start:
    mode = WORKING;
  completion: {
    main.Software.specifiedFeatures = 12;
    mode = STANDBY;
  }
  duration:
    return 3;
end
```

- **Dot notation:** `SoftwareDevelopment.Company.Alice.Specify` refers to the action `Specify` of the subsystem `Alice` of the subsystem `Company` of the system `SoftwareDevelopment`.
- **main** refers to the **root system**, here `SoftwareDevelopment`.
- `==` stands for equality test, `=` stands for assignment (of a value to a variable)
- As the **actor** is `Alice`, `mode` refers to the variable `mode` of `Alice`.

Activity: Code

```
activity SoftwareDevelopment.Company.Alice.Code
  trigger:
    return mode==STANDBY and main.Software.specifiedFeatures>=2;
  start: {
    main.Software.specifiedFeatures -= 2;
    mode = WORKING;
  }
  completion: {
    main.Software.codedFeatures += 2;
    mode = STANDBY;
  }
  duration:
    return 2;
end
```

- $X -= Y$; is a shorthand to $X = X - Y$;
- $X += Y$; is a shorthand to $X = X + Y$;
- Blocks of instructions are surrounded with curly braces.

Activity: Test

```
activity SoftwareDevelopment.Company.Bob.Test
  trigger:
    return mode==STANDBY and main.Software.codedFeatures>=4;
  start: {
    main.Software.codedFeatures -= 4;
    mode = WORKING;
  }
  completion: {
    main.Software.validatedFeatures += 4;
    mode = STANDBY;
  }
  duration:
    return 3;
end
```

Executions (1)

The underlying mathematical model of Σ^{TM} is **discrete event systems**. The semantics a S model is thus the set of all its possible **executions**.

In our example, the model is **deterministic** (as opposed as **stochastic**), consequently there is only one possible execution.

Initial state (time = 0):

Alice	Bob	Software		
mode	mode	specifiedFeatures	codedFeatures	validatedFeatures
STANDBY	STANDBY	0	0	0

In this state, the activity `Specify` of Alice is **enabled**, i.e. that its **trigger** returns true.

The two others are not enabled.

The action at start of the activity `Specify` of Alice is **scheduled** (at time 0).

As it is the only scheduled event, it is **fired** (still at time 0).

Executions (2)

Initial state (time 0):

Alice	Bob	Software		
mode	mode	specifiedFeatures	codedFeatures	validatedFeatures
STANDBY	STANDBY	0	0	0

After action at start of `SoftwareDevelopment.Company.Alice.Specify` (time 0)

Alice	Bob	Software		
mode	mode	specifiedFeatures	codedFeatures	validatedFeatures
WORKING	STANDBY	0	0	0

The action at completion of the activity `Specify` of `Alice` is scheduled at time $0+3=3$.

The other actions are still not enabled.

Executions (3)

After action at start of `SoftwareDevelopment.Company.Alice.Specify` (time 0)

Alice	Bob	Software		
mode	mode	specifiedFeatures	codedFeatures	validatedFeatures
WORKING	STANDBY	0	0	0

After action at completion of `SoftwareDevelopment.Company.Alice.Specify` (time 3)

Alice	Bob	Software		
mode	mode	specifiedFeatures	codedFeatures	validatedFeatures
STANDBY	STANDBY	12	0	0

In this state, the activity `Code` of `Alice` gets enabled.

It is thus scheduled at time 2.

The other activities are not enabled.

And so on.

Executions (4)

After action at completion of `SoftwareDevelopment.Company.Alice.Specify` (time 3)

Alice	Bob	Software		
mode	mode	specifiedFeatures	codedFeatures	validatedFeatures
STANDBY	STANDBY	12	0	0

After action at start of `SoftwareDevelopment.Company.Alice.Code` (time 3)

Alice	Bob	Software		
mode	mode	specifiedFeatures	codedFeatures	validatedFeatures
WORKING	STANDBY	10	0	0

After action at completion of `SoftwareDevelopment.Company.Alice.Code` (time 3+2=5)

Alice	Bob	Software		
mode	mode	specifiedFeatures	codedFeatures	validatedFeatures
STANDBY	STANDBY	10	2	0

And so on.

Collective Exercise

Question 1. Design a Σ^{TM} model for the life-cycle.

Question 2. Execute by hand this model.

LECTURE 2. GETTING STARTED

SECTION 3. THE WORLDLAB™ WORKSHOP

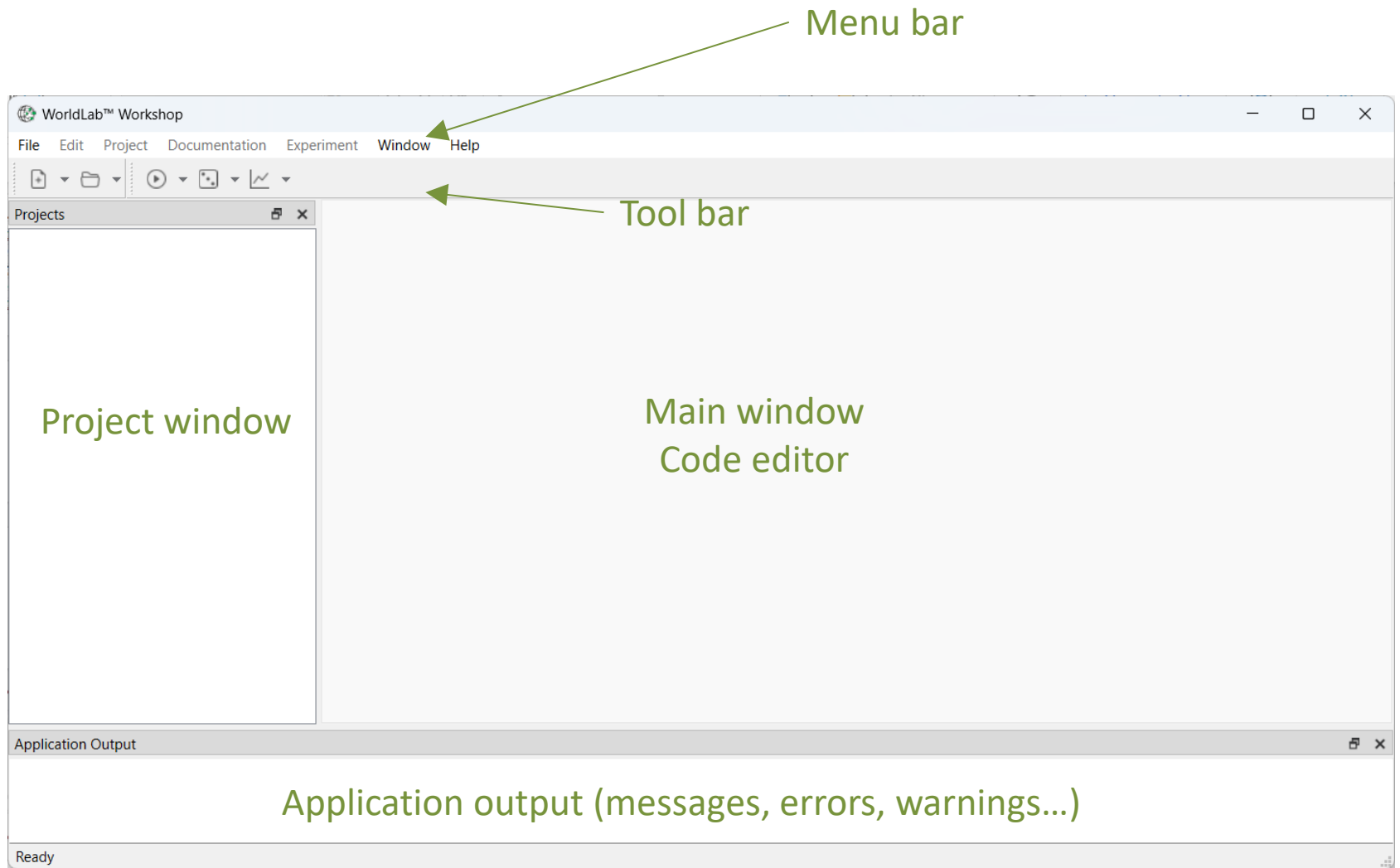
WorldLab™ Workshop

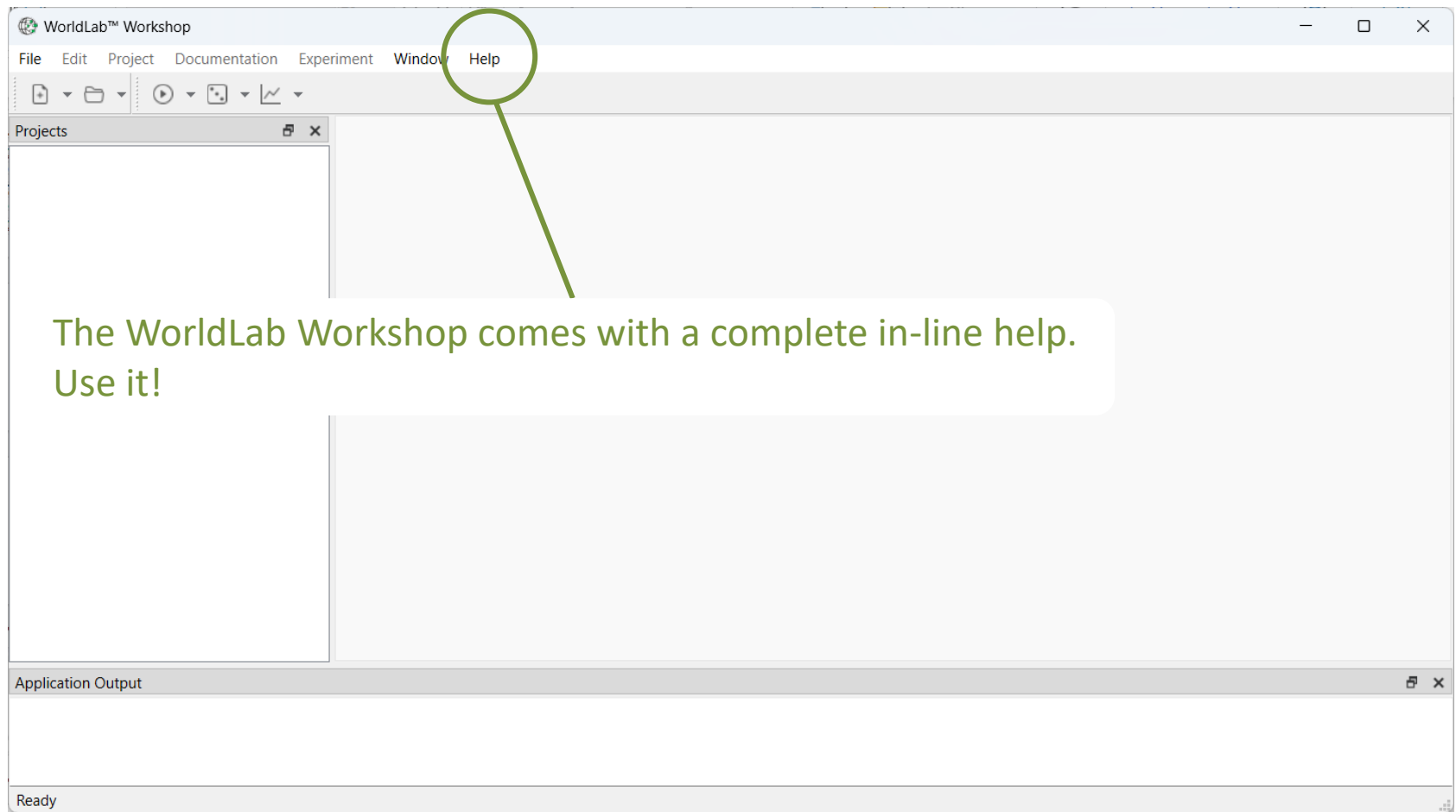
The **WorldLab™ Workshop** is an **integrated modeling and simulation environment** dedicated to the **Σ^{TM} modeling language**.

It is actually a **virtual laboratory** to design **systemic digital twins** of technical and socio-technical systems.

It embeds tools and features to:

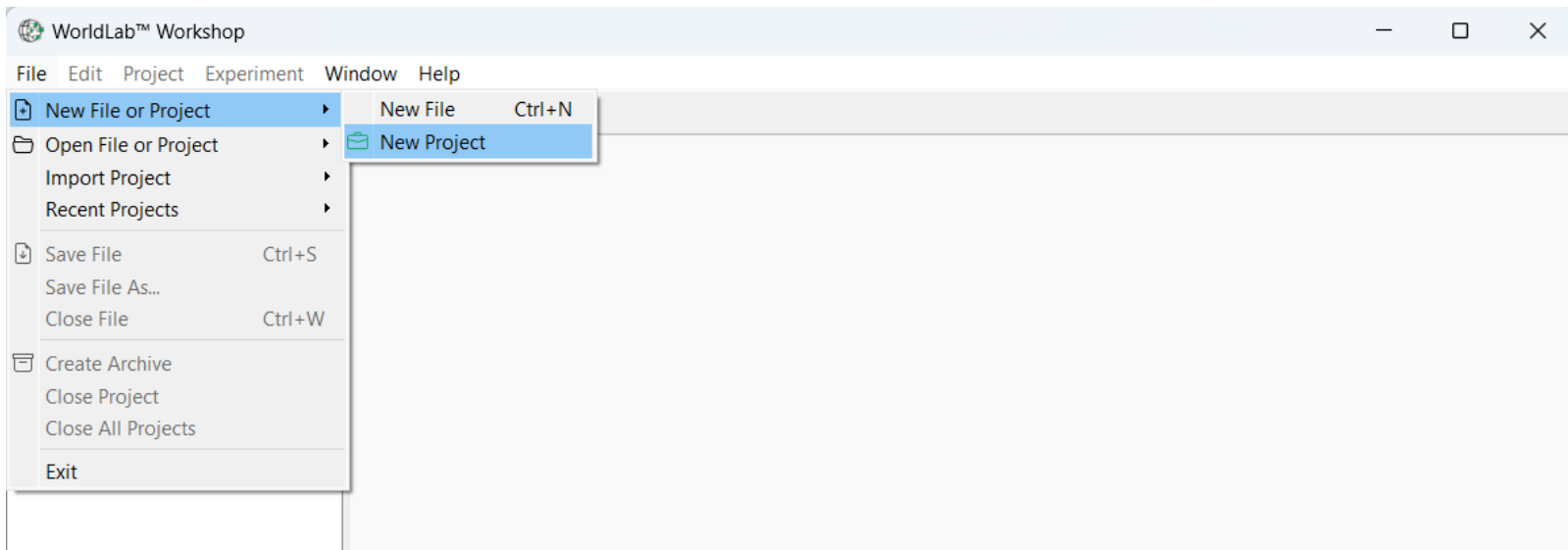
- Author **Σ^{TM} models**.
- Organize these models into **projects** and manages versions and configurations of these projects.
- Perform **interactive** and **stochastic simulations** on Σ^{TM} models.
- Record and replay **experiments** based on these simulations.
- Design dedicated **graphical user interfaces** for interactive simulations thanks to the **WIDL language**.
- Document projects and models thanks to the **Tau markup language** (similar to LaTeX).
- Create graphical model configurators using the **Kappa language**.
- ...





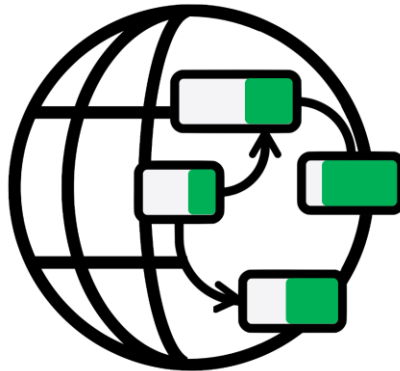
Projects

The WorldLab™ Workshop manages models (systemic digital twins) into projects. A **project** contains all the files the model consists off. It may also contain various documents as well as result files generated by simulators.



Create your projects in a specific folder.

Demonstration



LECTURE 2. GETTING STARTED

SECTION 4. INTERACTIVE SIMULATIONS

Rational

Interactive simulations make it possible to go forth and back in the execution of a model, while monitoring the values of variables and observers*.

Interactive simulations pursue actually two different objectives:

- **Check** models.
- **Share** models with stakeholders.

There are two ways of launching an interactive simulation in the WorldLab™ Workshop:

- By using the menu `Experiment>Interactive simulation`. Or,
- By clicking on the  button of the tool bar.

The Σ^{TM} Player is a graphical user interface to monitor interactive simulations.

(*) Observers will be presented next lecture.

Σ^{TM} Player

new* - SoftwareDevelopment - Sigma Player

File Simulation Window Help

Main System Schedule

Views

Component	Domain	Value	Unit
SoftwareDevelopment			
Company			
Alice			
mode	Mode	STANDBY	---
Bob			
mode	Mode	STANDBY	---
Software			
specifiedFeatures	int	0	---
codedFeatures	int	0	---
testedFeatures	int	0	---

Control board (video player metaphor)

Control Board

Seed 12345 Step 0 Time 0

Back Exit

Demonstration (SoftwareDevelopment1)



Collective Exercise

Question 1. Author your Σ^{TM} model for the life-cycle with the WorldLabTM Workshop.

Question 2. Perform an interactive simulation of this model with the Σ^{TM} Player.

LECTURE 2. GETTING STARTED

SECTION 5. DEDICATED SIMULATION INTERFACES

Rational

Interactive simulations are "the" mean to **communicate** with stakeholders about the model. It is thus of primary importance to make the interactive simulation interface as friendly as possible for non specialists.

The Σ^{TM} Player provides a default simulation interface. It is possible to design dedicated graphical user interface using the **WIDL language**.

WIDL is an object-oriented language to describe the views on the system at any step of an interactive simulation. It provides ready made views (essentially those of the default interface) as well as constructs to design your own views.

WIDL

```
block SoftwareDevelopment: Interface
  block Parameters: SigmaParameterTableView
    title = "Parameters";
    phases = [initialization];
  end
  block MainSystem: SigmaSystemListView
    title = "Project";
    phases = [initialization, simulation, reporting];
    filter = [systems, variables];
  end
  block Observers: SigmaObserverTableView
    title = "Observers";
    phases = [initialization, simulation, reporting];
  end
  block Schedule: SigmaScheduleTableView
    title = "Schedule";
    phases = [initialization, simulation];
  end
  block History: SigmaHistoryTableView
    title = "History";
    phases = [simulation, reporting];
    length = 20;
  end
end
```

Demonstration (SoftwareDevelopment2)



LECTURE 2. GETTING STARTED

SECTION 6. STOCHASTIC SIMULATIONS

Rational

Operation of complex technical systems and socio-technical systems is almost always subject to **uncertainties**.

In many cases, it is possible however to make assumptions on how these uncertainties are **statistically distributed** and to reflect these distributions in the model. The executions of the model are then **non-deterministic** (stochastic).

Monte-Carlo simulations consists in performing many executions and to perform statistics on the results.

Monte-Carlo simulations aim at assessing values of **key performance indicators**.

Demonstration (SoftwareDevelopment3)



LECTURE 2. GETTING STARTED

SECTION 7. DOCUMENTATION

Rational

A **systemic digital twin** consists not only of a model, but also of the **data** associated of this model, the **experiments** performed on the model and their **results**, and the more generally **all artifacts** involved in the design and the use of the model.

Moreover, the design of a model involves many **assumptions** that needs to be documented.

Regarding the documentation of a model (or a program), it is convenient to have it in two forms:

- As a **report**, typically a **PDF document** that can be printed out.
- As **HTML pages** that make it possible to navigate in the documentation.

Tau is a **markup language** that can be used to create both from the same source file.

The WorldLab™ Workshop embeds a Tau compiler that produces either HTML pages or a LaTeX document that can be subsequently compiled into PDF.

The HTML pages generated by the Tau compiler can be displayed directly in the WorldLab™ Workshop.

Tau

```
#+document
#+head
#title Software Development
#author Antoine Rauzy
#publisher NTNU
#version 4
#addBibliographyFile SoftwareDevelopment.bib
#-head
#+body
#makeFrontEnd

#section Introduction
The project (*code SoftwareDevelopment4*) aims at providing a glimpse on
how #:Sigma and the #:WorldLabWorkshop can be used to design (*em systemic
digital twins*) of complex technical and socio-technical systems.
...

#-body
#+document
```

Demonstration (SoftwareDevelopment4)



LECTURE 2. GETTING STARTED

SECTION 8. WRAP-UP

Wrap-Up

The Σ^{TM} **language** is a full-fledged object-oriented modeling language implementing ideas stemmed from both **system dynamics** and **discrete event systems**. The Σ^{TM} **ontology** relies on three pillars **systems**, **variables** and **activities**.

The **WorldLabTM Workshop** is an **integrated modeling and simulation environment** dedicated to the Σ^{TM} language. Beyond, it is a **digital laboratory** to design and use **systemic digital twins**.

It implements two main **experiments**:

- **Interactive simulations**, performed in the Σ^{TM} **player**, for which **dedicated graphical interface** using the **WIDL language**. Interactive simulations aim at debugging models as well as at discussing the model with the stakeholders.
- **Stochastic simulations**, performed in the Σ^{TM} **calculator**, that aim at assessing **key performance indicators**.

The WorldLabTM Workshop embeds also a compiler of the **Tau markup language** which can be used to document systemic digital twins.