# POP (Polychrony On Polarsys)

## A TOOLSET FOR SIGNAL

# POP Tool Design

V1.0

Tea team June 2015

# Table of Contents

1 Preface....3
1.1 Table of versions....3
1.2 Table of references and applicable documents....3
1.3 Acronyms and glossary....3
2 Subject....4
2.1 Purpose of the document....4
2.2 Editing particularities....4
2.2.1 Changes identification....4
2.2.2 Temporary editing....4
2.3 Application scope....4
2.4 Edition and evolution of the document....4
2.4.1 Responsibilities....4
2.4.2 Evolutions....5
3 Context and environment....5
3.1 SIGNAL language....5
3.2 Polychrony toolset....5
4 Justification of design choices....8
5 Architecture description....8
5.1 Static architecture....8
5.2 Dynamic architecture....10
6 Design description....11
6.1.1 The general architecture of the SIGNAL TOOLBOX software....11
6.1.2 The general architecture of the POP software....11
6.1.2.1 The SSME definition....11
6.1.2.2 Connection to the SIGNAL TOOLBOX....12
6.1.2.3 Compilation scenarios....12
6.1.2.4 Signal textual editor....12
6.1.2.5 Deployment of the POP Platform....12
6.1.2.6 Documentation....12
6.1.2.7 Help....13

# Index of Illustrations

# 1 Preface

## *1.1 Table of versions*

| ***Version*** | ***Date*** | ***Description & rationale of modifications*** | ***Sections modified*** |
|---|---|---|---|
| 1.0 | 15/04/2014 | First version | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

## *1.2 Table of references and applicable documents*

| ***Reference/ Applicable*** | ***Reference*** | ***Title & edition*** | ***Author or editor*** | ***Year*** |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

## *1.3 Acronyms and glossary*

| ***Term*** | ***Description*** |
|---|---|
| SSME | Signal Syntax Model under Eclipse |
| DCGraph | Data Control Graph |

# 2 Subject

## *2.1 Purpose of the document*

This document is the design documentation for the tool **POP (Polychrony On Polarsys).**

It defines the components of the tool and the dependencies, the API and the detailed design. It can also define the dynamic architecture if needed.

## *2.2 Editing particularities*

### 2.2.1 Changes identification

All the changes made since the previous publication are identified using the sign | in the left margin of each line holding a modification.

### 2.2.2 Temporary editing

Special points are signaled like this :

. ***temporary***

. ***incomplete***

. ***to be defined***

. ***to be confirmed***

## *2.3 Application scope*

This document is applicable for the tool **POP** that will be integrated in **Polarsys**.

## *2.4 Edition and evolution of the document*

### 2.4.1 Responsibilities

**Author**

The design document is written by members of the TEA Team (INRIA Rennes Bretagne Atlantique/IRISA).

**Checked by**

The design document will be checked by Loïc Besnard, Thierry Gautier and Paul Le Guernic.

**Approval**

The design document will be approved by Jean-Pierre Talpin.

**Diffusion**

The design document is provided to each member of the development team for application. It is also available on the forge with the source code of the tool.

### 2.4.2 Evolutions

The members of the TEA Team (INRIA Rennes Bretagne Atlantique/IRISA) are responsible of the evolution of the document.

This document shall be modified in case of change of the design of the tool.

# 3 Context and environment

The **POLYCHRONY TOOLSET**, is an Open Source development environment for critical/embedded systems. It is based on SIGNAL, a real-time polychronous dataflow language. It provides a unified model-driven environment to perform design exploration by using top-down and bottom-up design methodologies formally supported by design model transformations from specification to implementation and from synchrony to asynchrony. It can be included in heterogeneous design systems with various input formalisms and output languages.

## 3.1 SIGNAL language

SIGNAL is a specification and programming language for critical/real-time embedded applications. The main features of the SIGNAL languages are synchronized flows (flows + synchronization) and processes: a process is (recursively) a set of equations over synchronized flows describing both data and control. The SIGNAL formal model provides the capability to describe systems with several clocks (polychronous systems) as relational specifications. Relations are mandatory to write partial specifications, to specify non-deterministic devices (for instance a non-deterministic bus), and to abstract the behavior of external processes (for instance an unsafe car driver). Using SIGNAL allows to specify an application, to design an architecture, to refine detailed components downto RTOS or hardware description. The SIGNAL model supports a design methodology which goes from specification to implementation, from abstraction to concretization, from synchrony to asynchrony.

More details can be found in a short introduction to SIGNAL language on the Polychrony web site.

### 3.2 Polychrony toolset

The POLYCHRONY TOOLSET provides a formal framework:

- to validate a design at different levels, by the way of formal verification and/or simulation
- to refine descriptions in a top-down approach,
- to abstract properties needed for black-box composition,
- to assemble heterogeneous predefined components (bottom-up with COTS).
- to generate executable code for various architectures

The POLYCHRONY TOOLSET (See Illustration 1), contains three main components:

- The SIGNAL TOOLBOX, a batch compiler for the SIGNAL language, and a structured API that provides a set of program transformations. The SIGNAL

TOOLBOX can be installed without the other components.

ⓘ☞ ***The sources of the SignalToolBox (under GPL V2 license) are available on the*** Inria Polychrony web site***.***

- The SIGNAL GUI, a Graphical User Interface to the SIGNAL TOOLBOX (editor + interactive access to compiling functionalities). SIGNAL GUI requires the SIGNAL TOOLBOX ( or an other component that redefines the SIGNAL TOOLBOX APIs).

  ⓘ☞ ***The sources of the Signal GUI (under GPL V2 license) are available on the Inria Polychrony web site.***

- The POP PLATFORM, a front-end to the SIGNAL TOOLBOX in the Eclipse environment. POP PLATFORM requires the SIGNAL TOOLBOX (or an other component that redefines the SIGNAL TOOLBOX APIs). A meta model of Signal called SSME (Signal Syntax Model under Eclipse) is integrated in POP.

  ⓘ☞ ***The sources of the POP platform (under EPL license) are available on the pop-polarSys. ( Clone: https://git.polarsys.org/r/pop/org.polarsys.pop – branch SSME )***

The POLYCHRONY TOOLSET, also provides:

- libraries of SIGNAL programs,
- a set of SIGNAL programs examples,
- user oriented and implementation documentations,
- facilities to generate new versions.

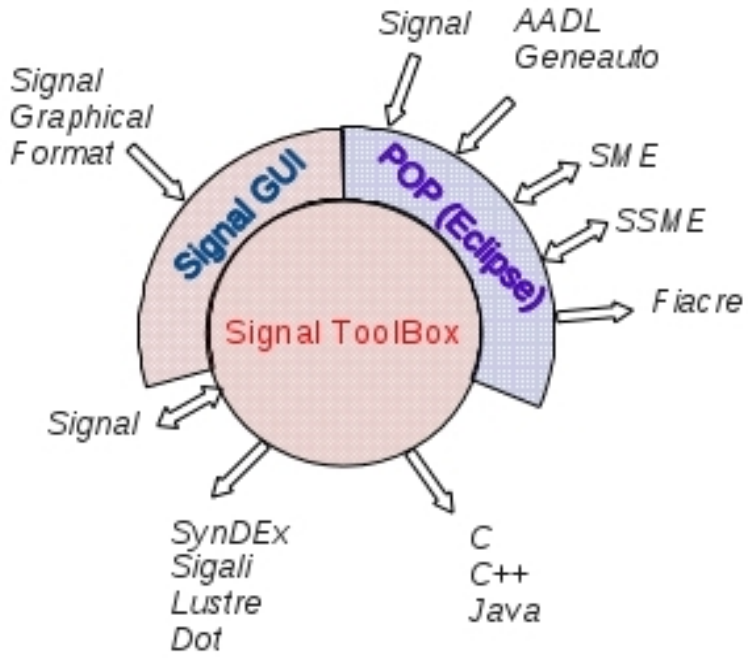


*Illustration 1: The Polychrony toolset*

The SSME model is used to import other formalisms (AADL, Geneauto/simulink,...) using Meta-Model to Meta-model translators (ATL, Kermeta, Java,...). These translators are not described here.

A SSME model as a Signal textual model may be compiled under the POP PLATFORM. For that POP is connected with the SIGNAL TOOLBOX. It allows to obtain some functionalities according to the compilation objective (see below). In the case of the batch compiler, the functionalities are provided through *options* (given in the command line). In the case of the SIGNAL GUI, they can be also applied interactively or in batch mode. And for the POP PLATFORM, the functionalities can be applied interactively, in batch mode, and also by specification of *compilation scenarios.*

Applying a compilation objective may modify the internal representation of a program (DCGraph: Data Control Graph). The result of the transformation can be rendered as a new Signal program. Such a graph is a representation of the program/system, at the different steps of its development, according to the Signal polychronous semantic model. Thus, it can be viewed/displayed by/to the user.

Other functionalities allow to produce code (sequential, distributed) for simulation (C, C++, Java) or for external tools such as SynDEx, Sigali (an associated formal system for formal

verification and controller synthesis), Lustre, Dot.

This document is reduced to the description of the POP platform and it connection with the SIGNAL TOOLBOX.

# 4 Justification of design choices

The SIGNAL TOOLBOX services have evolved over a continuous development and improvement research effort spanning since 1995, whereas the Eclipse Plug-ins date back since 2005. Consequently, the design of the POP PLATFORM was largely influenced by the SIGNAL TOOLBOX existing services.

In the following we will only delve into justification of the design choices at a high level view of the POLYCHRONY TOOLSET. Detailed information of the design choices (algorithms used, complexity measures, proofs of correctness, benchmarks, etc.) for the SIGNAL TOOLBOX services has been amply given in the form of research papers, technical reports, and PhD theses available from TEA team web site.

By contrast, detailed information about the design choices about the POP PLATFORM plug-ins will be omitted on the grounds that throughout most of the plug-ins development, automatic code generators were used, as will be noted below.

Many plug-ins of the POP platform are based on EMF Ecore meta-models. A reflexive editor has been generated using the EMF generation facilities from the SSME meta-model.

To allow to user to define its compiling scenario, a compilation scenario meta-model has been defined. POP provides the reflexive editor generated from this meta-model. An Eclipse view has been added to help the user to build a compilation scenario in a similar way as in the use of the classical SIGNAL GUI where one uses the interactive compilation.

The choice of using EMF has been made globally for all tools of the OpenEmbeDD and TopCased projects.

To access the Polychrony services under the Eclipse interface, the SIGNAL TOOLBOX has been tightly bound to the reflexive editor. The main goal for this connection is to obtain traceability between the SSME models and the results of a compilation. Thus, it becomes possible to indicate directly on the source model the compilation errors. Such connection already exists between the Polychrony SIGNAL TOOLBOX services and the Polychrony SIGNAL-GUI classical graphical editor.

# 5 Architecture description

## *5.1 Static architecture*

Here we give (see Erreur : source de la référence non trouvée ) a global view of the SIGNAL TOOLBOX environment. Then, we explain the connection of the SIGNAL TOOLBOX to the Eclipse plug-ins:

- A program represented as an Abstract Tree is analyzed (the form is verified: declarations, references to models are solved, etc.) and reduced to a “subset of **Signal**”. The resulting object can be seen as a **Signal** program without “syntactic

sugar” of the textual form.

- Then a *DCGraph* is produced from this “reduced form”: this graph represents the synchronizations of the program (control part) and the calculus of the program (data part).
- It is on this graph that the functionalities are applied:
  - Application of a functionality can modify the graph. In this case, it results in a new Signal program. Such a graph is the representation of a model, at the different steps of its development, in the Signal polychronous semantic model. So, it can be displayed to the user (textual form, SSME model...).
  - Other functionalities allow to produce code for simulation (C, C++, Java) or for external tools (SynDEx, Sigali...).

The Signal Abtract Trees transformations (SynAPI), the production of the DCGraph (SynSemAPI) and the functionalities applied on a DCGraph (SemAPI) are provided by the Signal ToolBox for external tools (Signal GUI, POP Platform,...).

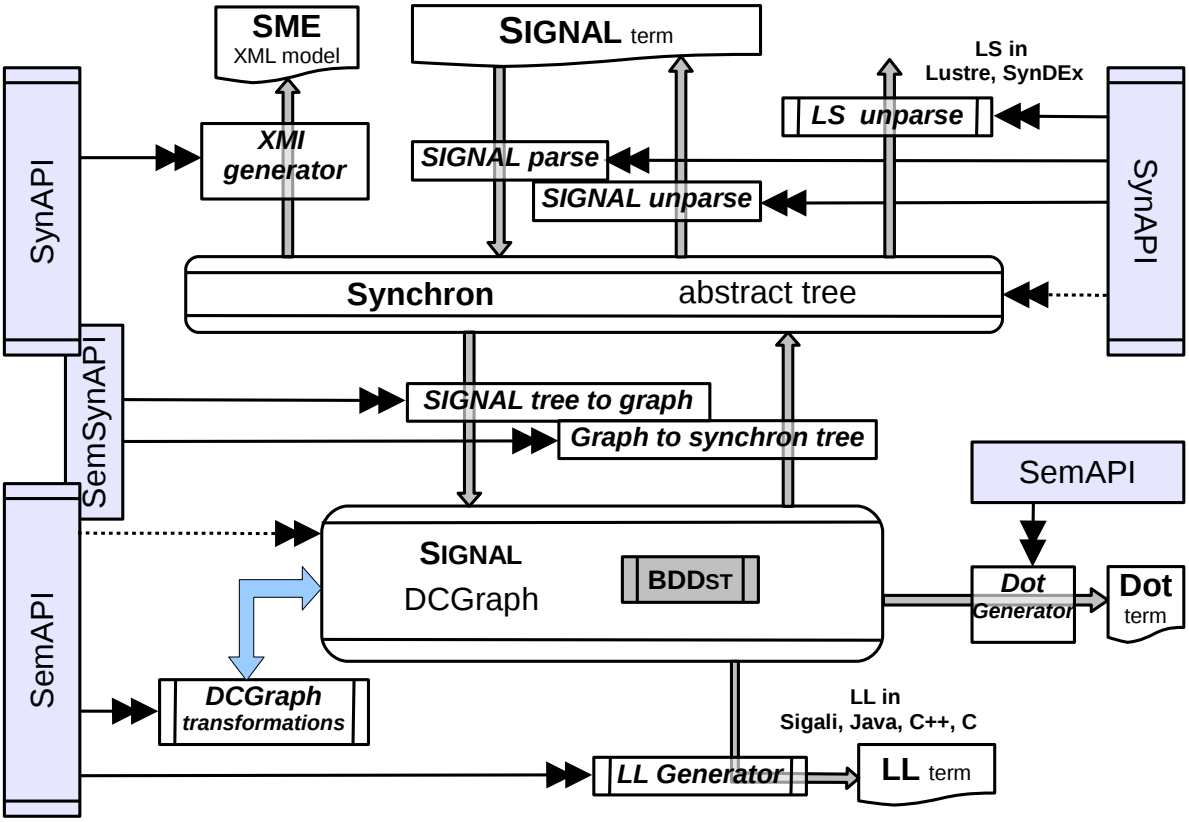


*Illustration 2: The Signal ToolBox high level architecture*

**The connection to the Signal Toolbox services**

The connection between the SSME Eclipse editors and the SIGNAL TOOLBOX consists in a Java-to-C interface to communicate with the SIGNAL TOOLBOX through native libraries. The principle of the communication is the following: the SSME model is transformed into an Signal abstract syntax tree representation shared with the SIGNAL TOOLBOX (using the SynAPI provided by the Signal ToolBox), then, the steps described above are applied to this abstract tree (using the SynSemAPI and the SemAPI).

## *5.2 Dynamic architecture*

The chief internal representation for the SIGNAL TOOLBOX services is the *DCGraph* mentioned above. The DCGraph is modified by application of each functionality. Roughly, all possible dynamic architecture descriptions of this part of the Polychrony tool correspond to a specific path through the static architecture described above. Such a specific path must agree with the compilation objective demanded.

The information communicated between reflexive editors, GUI and compilation scenarios is through XMI files containing information conforming to a reference meta-model (SSME meta-model and configuration files). A description of the dynamic architecture details of standard tools (EMF and graphical edifor generators) for generation of the various editors is beyond the scope of the TEA team.

# 6 Design description

### 6.1.1 The general architecture of the SIGNAL TOOLBOX software

The design of the Polychrony SIGNAL ToolBox is described in the document provided with the SIGNAL TOOLBOX.

The SemAPI and the SynAPI provided by the SIGNAL ToolBox are used by the POP PLATFORM.

### 6.1.2 The general architecture of the POP software

The POP software is composed of the following parts:

- the plug-ins that define the SSME meta-model and the associated reflexive editor
- the plug-ins that define the connection to the SIGNAL TOOLBOX.
- the plug-ins that define the compilation scenarios of a SSME model or Signal Textual model.
- The plug-ins that define a Signal textual editor under Eclipse.
- The features used for the deployment
- The plug-in that define the update site
- The documentation plug-in
- The help plug-in

#### 6.1.2.1 The SSME definition

As mentioned previously, the structure of POP is based on the EMF and automatic generators. The integration of POP in Eclipse is composed of:

A reflexive editor to manage SSME models (conform to the SSME meta-model). It is composed of the following plug-ins:

- **org.eclipse.pop.ssme**: contains the definition of the SSME model (ssme.ecore), the Java classes (interface/implementation), which represent each concept of the SSME meta-model and the way they are serialized in XMI. The construction of the native AST (used for the connection to the SIGNAL TOOLBOX) is done by these classes. The “model” directory of this plug-in contains the SSME meta-model.
- **org.eclipse.pop.ssme.edit**: contains the Java classes which manage the connection between the model objects and the Eclipse property view.
- **org.eclipse.pop.ssme.editor**: contains the Java classes for the SSME reflexive editor and for the SSME wizard to create new models.

### 6.1.2.2 Connection to the SIGNAL TOOLBOX

The connection to the SIGNAL TOOLBOX is composed of one plug-in:

- **org.eclipse.pop.ssme.polychrony** is the plug-in, which contains the Java classes of the JNI interface corresponding to the call to Polychrony SIGNAL TOOLBOX services.

### 6.1.2.3 Compilation scenarios

A reflexive editor to manage SSME compilation scenario models has been developed. It is composed of the following plug-ins:

- **org.eclipse.pop.ssme.compilation, org.eclipse.pop.ssme.compilation.edit, org.eclipse.pop.ssme.compilation.editor**: are the equivalent of the previously presented plug-ins for the SSME reflexive editor. The “model” directory of the first plug-in here contains the SSME compilation scenario meta-model.
- **org.eclipse.pop.ssme.compilation.utils**: contains the Java classes for the compilation scenario view that guides the user through the definition of correct compilation scenarios. It also defines all actions that are available to the user to call SIGNAL TOOLBOX native services.

### 6.1.2.4 Signal textual editor

A Signal textual editor under Eclipse (Signal mode) is also provided in a plug-in called **org.eclipse.pop.ssme.texteditor.** It is used when a file suffixed by sig or SIG is edited.

### 6.1.2.5 Deployment of the POP Platform

The following features and plug-ins are defined to allow to deploy POP:

- **org.eclipse.pop.ssme.feature**: the reflexive editor + the documentation + the connection to the native compiler (assuming dependencies of this feature have already been properly installed/resolved/deployed).
- **org.eclipse.pop.ssme.compilation.feature:** The reflexive editor for the compilation scenario models + the access to native compiler services + the documentation.
- **org.eclipse.pop.ssme.texteditor.feature:** The signal textual editor.
- **org.eclipse.pop.ssme.site:** plug-in that manages the production of the POP update site.

### 6.1.2.6 Documentation

The documentation of the SSME environment, the connection to the SIGNAL TOOLBOX and the installation guide are managed in a plug-in called

- **org.eclipse.pop.ssme.documentation.**

The documents are generated from OpenOffice documents.

The documentation of the sources are generated (using the Eclipse “Project->generate javadoc” action) in the **doc** directory of the following projects:

- **org.eclipse.pop.ssme** : for the SSME meta-model, the reflexive editor and the connection to the SIGNAL-TOOLBOX.
- **org.eclipse.pop.ssme.compilation** for the reflexive editor to manage SSME compilation scenario
- **org.eclipse.pop.ssme.texteditor** for the Signal textual editor under Eclipse

### 6.1.2.7 Help

The help of the POP platform is managed in a plug-in called

**org.eclipse.pop.ssme.help.**

The help is generated from doxygen documents (using the “build documentation” action of the Eclipse menu).