POP

A TOOLSET FOR SIGNAL

POP Specification

V1.0

	Author(s)	Checked by	Approval
Name	**Members of the TEA Team**	**Loïc Besnard **** **Thierry Gautier** **Paul Le Guernic**	**Jean-Pierre Talpin**
Company	**INRIA**	**INRIA, **CNRS (SED)**	**INRIA**
Department	**TEA Team**	**TEA Team**	**TEA Team**
Date	**April 2014**	**April 2014**	**April 2014**
Visa			
Summary	**Specification of the Polychrony SSME, a front-end to the Signal ToolBox in the Eclipse environment.**		

Table of Contents

Index of Illustrations

1 Preface

1.1 Table of versions

Version	Date	Description & rationale of modifications	Sections modified

1.2 Table of references and applicable documents

Reference/ Applicable	Reference	Title & edition	Author or editor	Year

1.3 Acronyms and glossary

Term	Description
SSMESSME	Signal Syntax meta-model under Eclipse

2 Subject

2.1 Purpose of the document

This document is the specification for the tool POP (Polychrony on Polarsys)

It defines the requirements for the tool and the performances.

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 specification document is written by the members of the TEA Team (INRIA/IRISA, Rennes).

Checked by

The specification document will be checked by Loïc Besnard and Thierry Gautier.

Approval

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

Diffusion

The specification 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/IRISA, Rennes) are in charge of the evolution of the document.

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

3 Context and environment

The Polychrony plug-ins under Eclipse (called POP platform) allow to develop applications using the Signal language. The **Signal** language, a polychronous data-flow language, and the **Polychrony** toolset for Signal have been designed at IRISA. The **Plug-in suite** is composed of several plug-ins which correspond to:

- the reflexive editor,
- the reflexive editor and an Eclipse view to create *compilation scenarios*,
- the connection to the Polychrony services (called Signal ToolBox). A service can be applied or not according to the objectives of the compiling defined in the compilation scenarios.

The reflexive editor is the plug-in generated by the Eclipse Modeling Framework (EMF) from the meta-model of Signal, referred to as the SSME meta-model (Signal Syntax meta model under Eclipse). It allows to specify a SSME model by creating the equivalence of an abstract syntax tree where the syntax is given by the meta-model. SSME may be also used as a pivot language for importing other formalisms (AADL, Geneauto/simulink,...).

The compiling of a Signal textual model or a SSME model is the application of some functionalities (optimizations, code generation, formal verifications...). provided by the external (to POP) Signal ToolBox. These functionalities are proposed in interactive compiling mode, for the definition of a scenario compilation, and also in batch mode compiling by options. A functionality modifies the internal representation of a Signal program (it is a new Signal program, that can be visualized using the modeler) whereas a generator translates this representation into a specific external format (C, C++, Java...). During the “compiling” a trace is returned to the user in a specific window.

4 Functional requirements

As shown before, the plug-ins allow to edit, and compile Signal programs using the SSME model or Signal Textual form. In this section, the functional requirements are listed. Each requirement is tagged with R_ToolName_Category_nnn in order to facilitate the traceability, with

- R = Requirement,
- ToolName = the name or an acronym of the tool (POP),
- Category = the name of a coherent group of functionalities. We distinguish the two categories “editing” and “compiling”.
- nnn = a number on 3 digits.

4.1 Editing

****** to be defined******

4.2 Compiling

The functionalities the user can apply on a Signal/SSME model are composed of the

- transformations : the Signal program is rewritten in an another program by applications of some rules.

- export tools: the Signal program is exported for some external tools,
- code generators: the Signal program is exported in programming language (C, C++,Java)
- export Signal: the Signal program is exported as a new program Signal.

For more details about how to apply these functionalities, see the POP User Guide provided in the Eclipse help (**Help->Help Contents->POP documentation**).

In the following rules, we use the Signal syntax; for more information about the semantics of each element, consult the Signal v4 reference manual.

These functionalities are proposed in interactive compiling mode, for the definition of a scenario compilation, and also available in batch mode compiling using options (given in parenthesis).

Transformations

- **/R_POP_compiling_001/ Retiming** (-dr): performs an UPWARD normalization of delayed signals. It rewrites synchronous function f such that Y := f(X1 $ m1 init V1,...,Xn $ mn init Vm) into Y := y' $ j init f(V1',...,Vm') and y' := f(X1 $ m1' init V1",...., Xm $ mn' init Vm").
- **/R_POP_compiling_002/ Booleans to events** (-crew): performs an event normalization of boolean clock expressions (example: when (a and b) -> when a when b).
- **/R_POP_compiling_003/ Signal unifications** (-su): performs a signal syntactic equivalence reduction. For example, for (| x:=E | y :=E |) x is replaced by y.
- **/R_POP_compiling_004/ Clock calculus** (-poly): performs the resolution of the clock systems using a triangularization technique. The result is a forest of clock trees (hierarchy with several roots).
- **/R_POP_compiling_005/ Endochronisation**(-endo): reduces the hierarchy with several roots (polychronous model) to one root (endochronous model).
- **/R_POP_compiling_006/ Events to booleans** (-bool) : produces an endochronous model without "events".
- **/R_POP_compiling_007/ Sequential clustering** (-clu) : performs a code separation with respect to input predecessors equivalence.
- **/R_POP_compiling_008/ Abstraction** (-spec): computes the abstraction of the program (I/O data dependences, I/O clock relations, the "black Box" or the "grey Box" abstraction representation).
- **/R_POP_compiling_009/ Sequentializing** (-seq): produces a SIGNAL sequential code (reinforcing of the dependencies)
- **/R_POP_compiling_010/ Flattening** (-flat): produces the SIGNAL code in which the hierarchy of clocks is reduced (flattened) to one level.

Export tools

- /R_POP_compiling_011/ **Sigali**: Generates code (.z3z files) for Sigali tool, used to prove dynamical properties.
- /R_POP_compiling_012/ **Lustre**: Generates lustre code (.lus file).
- /R_POP_compiling_013/ **Syndex**: Generates code (.sdx file) for SynDEx tool, used for code distribution.

Code generators

- /R_POP_compiling_014/ **C ANSI**: Generates C code (.c and .h files), with clusters if the Sequential clustering ha been applied.
- /R_POP_compiling_015/ **C++**: Generates C++ code (.cpp and .h files), with clusters if the “Sequential clustering” has been applied.
- /R_POP_compiling_016/ **Java**: Generates Java code (.java) with clusters if the “Sequential clustering” has been applied.

Export Signal

- /R_POP_compiling_017/ **Signal Textual**: produces the textual Signal file for a SSME model.
- /R_POP_compiling_018/ **Signal Model (SSME)**: produces the SSME form for a textual Signal file.
- /R_POP_compiling_019/ **Signal Textual (LIS)**: produces a Signal file containing the pretty printed definition.
- /R_POP_compiling_020/ **Signal Textual (TRA)**: produces a Signal file containing the result of the applied transformations.
- /R_POP_compiling_021/ **Signal Abstraction**: produces a Signal file containing the abstraction of the compiled model (I/O clock relations, I/O dependences, ...).
- /R_POP_compiling_022/ **Profiling**: produces a morphism of the compiled program for profiling. The path of the morphism table must be in the SIGNAL_LIBRARY_PATH shell variable.

4.3 Scenarios compiling

****** to be defined******

5 Operational environment requirements

******to be confirmed******

Software requirements follow:

- (R_POP_SoftwareEnvironment_001) JRE: Oracle Java JRE 1.6

- (**R_POP_SoftwareEnvironment_002**) Eclipse: Eclipse Modeling 4.3 (Kepler)
- (**R_POP_SoftwareEnvironment_003**) C/C++ compiler: If compilation of the generated C/C++ programs is desired for performing simulations the user may have a C/C++ compiler.
- (**R_POP_SoftwareEnvironment_004**) Java compiler: If compilation of the generated Java programs is desired for performing simulations the user may have a Java compiler.
- (**R_POP_SoftwareEnvironment_005**). Explicit dependencies are shown in the Polychrony SCP document.

There is no special hardware requirement for using the POP tool, we assume that a standard machine in the market won't pose problems for the processing required by the POP tool. The most heavy program is not the POP tool but the Eclipse environment.

- (**R_POP_HardwareEnviroment_101**) Processor: Our development and experiments are run under PC/laptop machines with Pentium 4, Core Duo or Core 2 Duo processors. Anything similar or better would suffice.
- (**R_POP_HardwareEnvironment_102**) RAM memory: 1Gigabyte minimum, given the demands of Eclipse.
- (**R_POP_HardwareEnvironment_103**) Hard disk free space: For the binary distribution it is required 12Mb of free disk space. The binaries with the sources demand 95Mb of free disk space.

6 Interface requirements

******incomplete***

It is a set of plug-ins under Eclipse, so Eclipse must be available and running.

7 Performance requirements

This is so subjective, it depends on the machine (hardware) and the concurrent tasks executing at the time of launching Eclipse.

Also the environment configuration of Eclipse shall count.

8 Traceability

The effect of the application of a functionality during the “compiling” step must be returned to the user

(textual form or graphical form).